The Report Committee for Vipul Goyal
Certifies that this is the approved version of the following report:

Physical Design and Verification for Microscale Modular Assembled ASIC (M2A2) Circuits

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: S. V. Sreenivasan

Co-Supervisor: Mark McDermott

Physical Design and Verification for Microscale Modular Assembled ASIC (M2A2) Circuits

by

Vipul Goyal, B.Tech

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2017

Acknowledgements

I would like to thank my supervisor, Prof. S. V. Sreenivasan for giving me this great opportunity. His guidance, encouragement helped me a lot in this endeavor. I would like to express my sincere gratitude to Prof. Mark McDermott for being always available for technical discussion and providing feedback. Without his help, this report would not have been possible.

I would also like to thank project partners, Aseem Sayal, (E.C.E. Graduate Student, UT Austin) and Paras Ajay (M.E. Graduate Student, UT Austin) for all their valuable inputs. Also, I highly appreciate the effort of Shrawan Singhal (Post-Doctoral Research Fellow, M.E., UT Austin) for helping me with writing the report. I would like to extend my sincere gratitude to Nandini D Chintala, Prashant Joshi, Bhargav Bhat Bhaskar and Cadence team for giving their valuable suggestions and guidance.

Also, I'm thankful to The University of Texas at Austin for providing the required infrastructure for this project. A special thanks to all the friends and people in industry who helped me to understand the ASIC tools and flows.

Abstract

Physical Design and Verification for Microscale Modular Assembled ASIC (M2A2) Circuits

Vipul Goyal, M.S.E.

The University of Texas at Austin, 2017

Supervisors: Prof. S. V. Sreenivasan, Prof. Mark McDermott

The overall goal of this project is to bring down the fabrication cost for low volume ASICs by introducing a novel 'pick and place' mechanism for micro-scale elements of ASICs referred to here as feedstock. This new feedstock based ASIC design flow is referred as Microscale Modular Assembled ASIC (M2A2) design flow. This report complements efforts in fabrication and other Electronic Design Automation (EDA) aspects carried out by researchers at The University of Texas at Austin studying this new mechanism for ASIC design and manufacture. For the purpose of this study, the conventional industrial practice in ASIC design flow was analyzed and modifications to that flow were explored. The initial Synthesis solution was developed using Synopsys's Design Compiler (DC) tool. However, due to the limitations of the tool, the final solution was developed based on Cadence tools. The main blocks of the design flow in this report are Synthesis and analysis of its capabilities; Conformal ECO; Post-Mask spare cell mapping; Post-Mask Clock Tree Synthesis (CTS) and Route; Post-Mask timing and

Design Rule Violation (DRV) fixing; and Verification. The Standard Cell-based ASIC design was used as a benchmark and it was compared to M2A2 design flow.

Table of Contents

List of Tables

List of Figures

Introduction

In the world of ASICs, the fabrication cost has gone up with technology scaling [1]. This is because the cost of e-beam lithography, which is used to generate masks, grows exponentially with diminishing feature size, especially for sub-50 nm technology nodes. This has a significant impact on low volume ASICs. Since their chip-count is typically in the hundreds or thousands, the fabrication cost per IC can be very high. The aim of this project is to bring down the fabrication cost of low volume ASICs without compromising on performance. Some existing solutions for low volume ASICs include structured ASIC [2,3,4,5,6,7,8] and FPGA-based flows. Although a FPGA-based flow is reconfigurable and has very less time to market, it consumes a lot of area, and suffers from low performance with high costs. On the other hand, structured ASICs have challenges with implementing a general design without negatively impacting area, performance, and power. Also, Via-programmable ASIC [9-10] have been tried but it has its limitation. As part of this project, we have added an extra degree of freedom to the structured ASIC flow without compromising performance, by coming up with an innovative 'pick and place' mechanism [11] of 'feedstocks' on the target SOC, for fabrication of low volume ASICs. These feedstocks are the fundamental building blocks of the ASIC design.

However, the current standard ASIC design flow needs to be modified to support feedstock-based fabrication. The first step is to design the feedstock library which will serve as the fundamental building blocks in the design. The next step is to take the

customer's design, i.e. the Register Transfer Logic (RTL) (Figure 1) and run the standard cell-based flow to meet the design requirements (Figure 2).

```
if (sel)
    y= b
else
    y = a
```

Figure 1: Sample customer's RTL

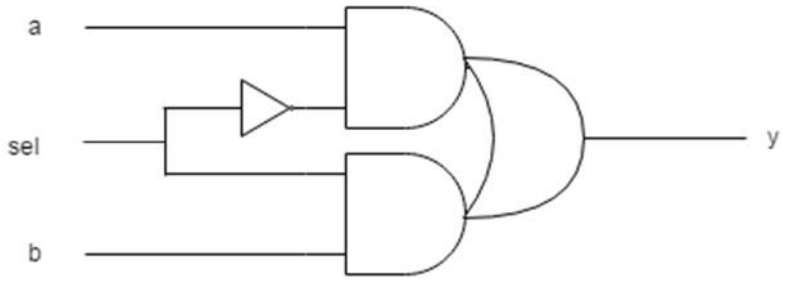


Figure 2: Standard Cell based design

The standard cell-based design is divided into an array of windows. A scanning-based approach is adopted, wherein for each window, the best fit feedstock from the feedstock library (Figure 3) is assigned based on the physical location of cells. After the windows have been mapped to different feedstocks, the base layer is fixed and the feedstock level netlist (Figure 4) is generated. This generated netlist might have partial connectivity.

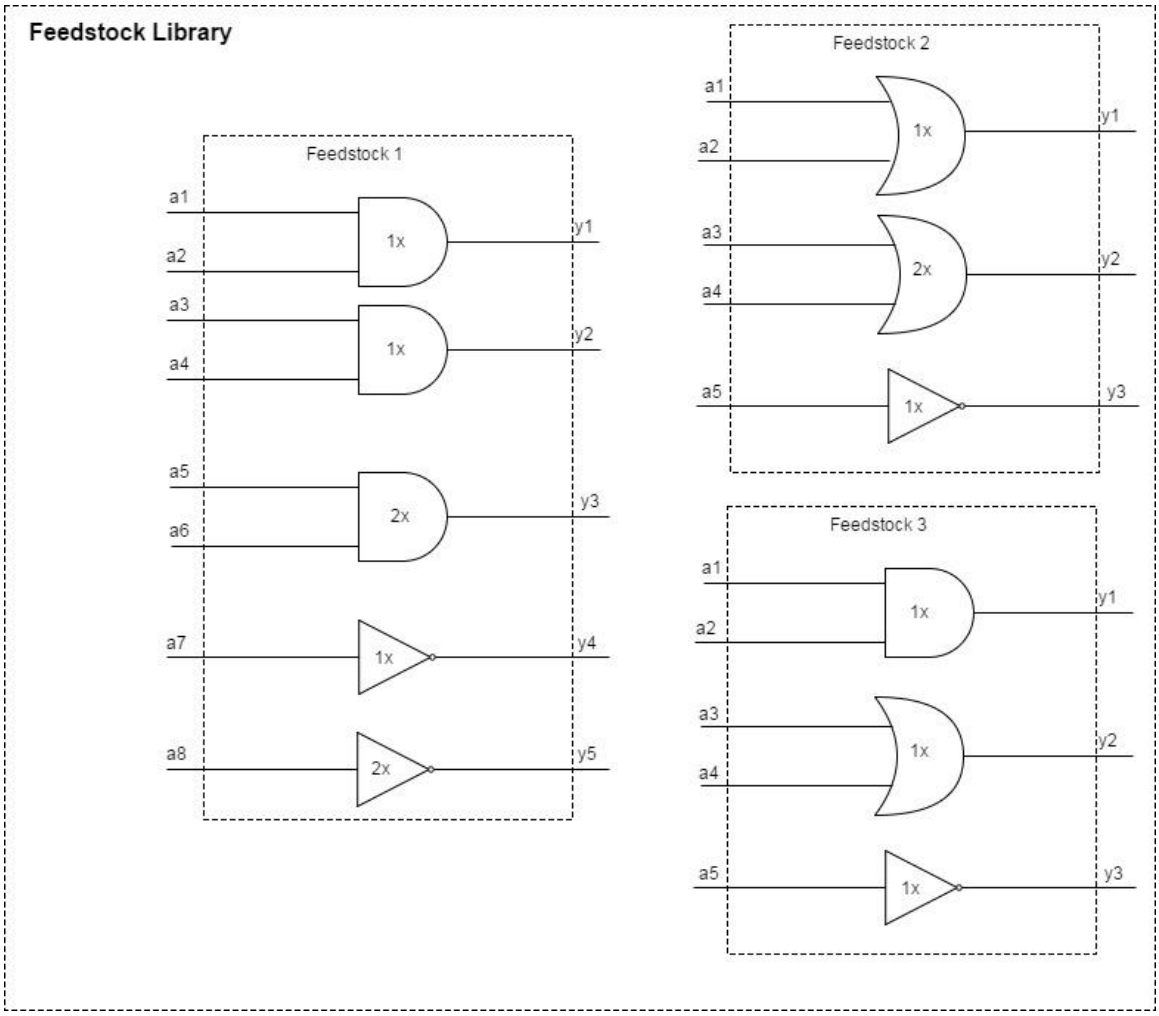


Figure 3: Library of Feedstocks

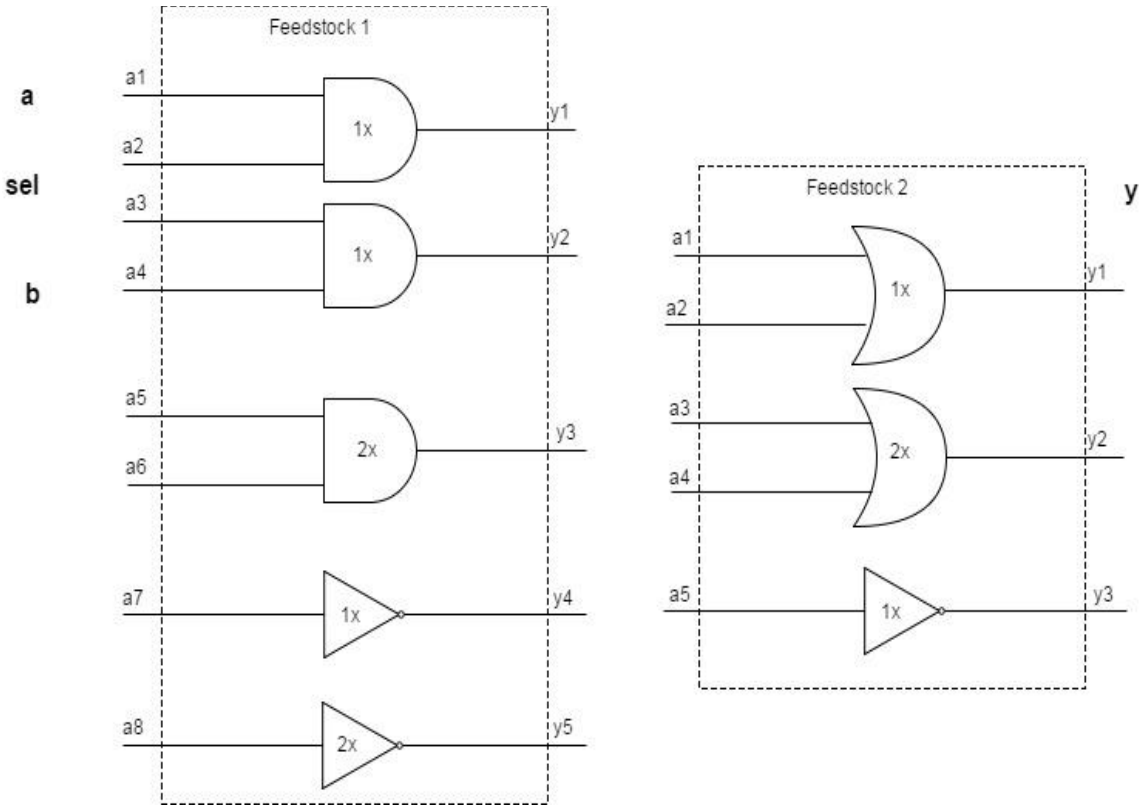


Figure 4: Feedstock based netlist which will be used in Conformal ECO

The next step in the design flow is to use Conformal ECO and import the feedstock-based netlist (Figure 4) as the golden netlist, and the standard cell-based netlist (Figure 2) as the revised netlist. In this step, all the cells in the design are defined as spare cells. However, if there is partial connectivity in the golden netlist, then only the leftover cells are defined as spare cells. After defining the spare cells, Conformal ECO attempts to simultaneously meet the functionality (Figure 5) and optimize the design. Then, the swap spare cell file is generated with one-to-one mapping between each spare cell and its corresponding usage in the design.

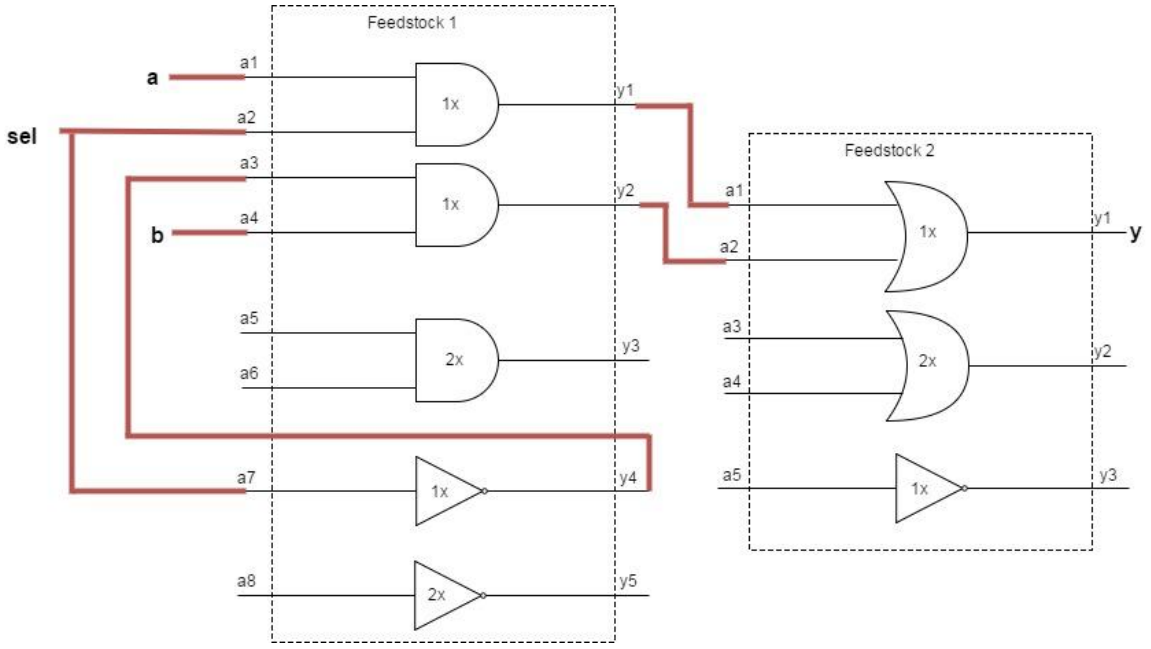


Figure 5: Getting the functionality using Conformal ECO

After the generation of the spare cell mapping, PostMask Place (physically mapping the spare cells) and Route (P&R) is used to route the design to get the same functionality using only spare cells. Next, a timing analysis is done to fix the timing numbers that are compromised due to the interconnect delays. If there is any negative slack, the timing and DRV violations are fixed using Tempus and Innovus. In this step, it is ensured that the base layer is not changed and that only the spare cells are used to optimize the design (Figure 6). A timing analysis is done again to check if the timing goals are met. This process is reiterated with a different set of feedstocks if, after this step, the design is not closed.

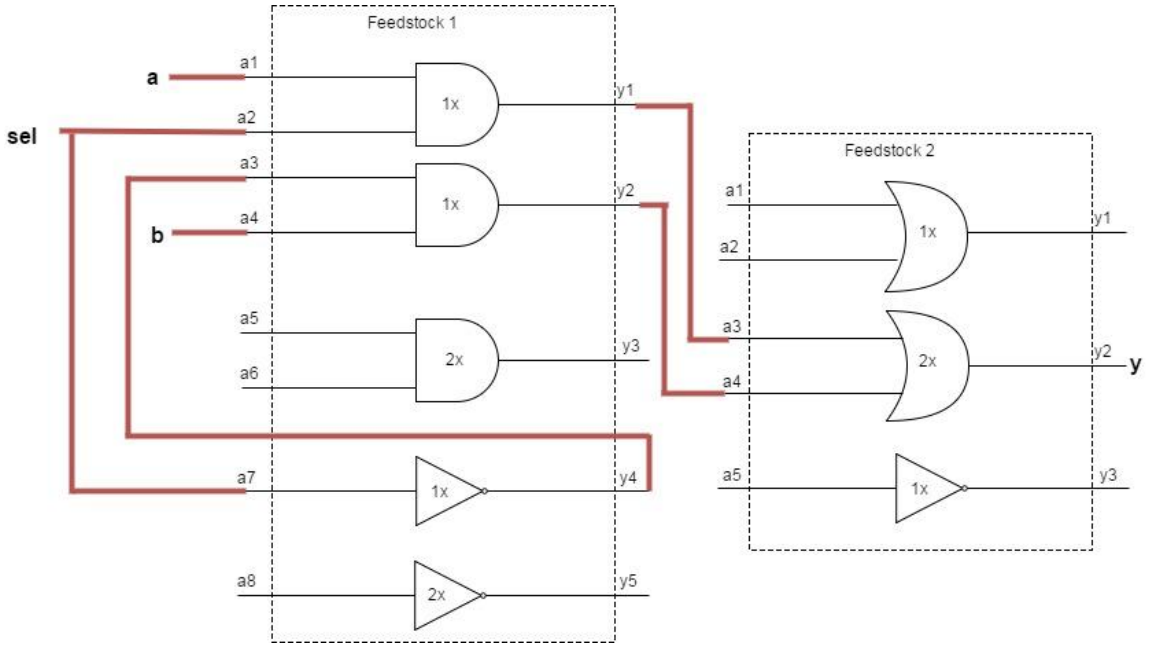
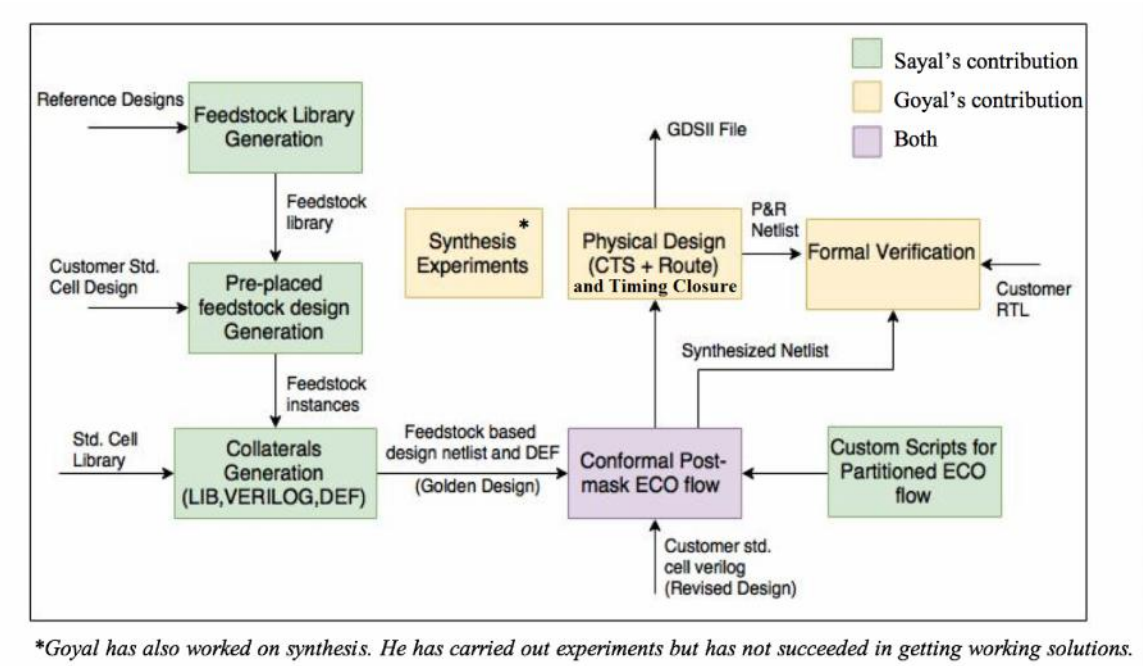


Figure 6: Optimizing the design

Finally, after the design requirements are met, verification checks are done to compare the functionality of the final design with respect to the customer's design with a provision to also compare the feedstock based netlist against the original RTL.

The M2A2 design flow has been collaboratively developed with Paras Ajay [11] and Aseem Sayal [12]. Ajay has developed the 'pick and place' mechanism to fabricate the feedstock based SOC. Sayal and I have worked on the Electronic Design Automation (EDA) design flow with the division of work shown in Figure 7. Sayal has worked on the feedstock library generation, pre-placement of the feedstocks and generation of the collaterals. He has also worked on custom scripts for partitioned ECO flow. The Conformal Post-Mask ECO flow has been done jointly. In this report, the physical

implementation has been conducted by mapping the physical locations of the spare cells to the Conformal netlist; Post-Mask CTS and Routing; Post-Mask DRV and Timing closure. Finally, Formal Verification is done to compare the functionality of the P&R netlist with the customer RTL



**Goyal has also worked on synthesis. He has carried out experiments but has not succeeded in getting working solutions.*

Figure 7: Work Distribution [13]

Chapter 1: Synthesis

In the Synthesis stage of the design, the customer RTL is mapped to the fundamental blocks, called standard cells. As part of this project, it is required that these standard cells be bundled together in a single feedstock. Each feedstock can have different numbers and flavors of standard cells. Because of the pick-and-place fabrication approach, the use of any standard cell from a given feedstock requires the pick-and-place of the entire feedstock. Hence, an important optimization constraint in the Synthesis stage is the optimal use of all the cells in the chosen feedstock before adding any new feedstock in the design.

In this chapter, various experiments that were carried out on existing standard Synthesis tools [14] to check if they could meet the project's requirements, are discussed. An analysis of the current design flow is presented along with modification and limitations. While the Synthesis flow was initially developed using the DC tool of Synopsys [15] as seen in the following sections, it was later developed based on Cadence tools, given the limitations of the Synopsys tool.

1.1 Defining each feedstock as a 'Standard' cell

The first approach was to define each feedstock as a cell in the library. This approach made use of the standard flow, where a given cell could be picked any number of times based on the requirements of the design. Similarly, different feedstocks could be defined as different cells in the library. A variety of experiments were performed to see if the tool understood these feedstocks as cells.

1.2 EXPERIMENTS TO DEFINE FEEDSTOCK AS 'STANDARD CELL'

As mentioned earlier, experiments were carried out to check if the Synthesis tool supported 'standard cells' with multiple independent designs. As a starting point, the synthesis of a small design was tried using these modified 'standard cells'. A normal synthesis flow was first setup with the required collaterals like lef file, tech file, and constraints being the same as in a standard cell-based design. For each experiment, a modified .lib was generated. The standard cells library was replaced with this modified library.

The basic format of a Synopsys library [16] is shown below –

```
library (<library_name>) {
     cell (<cell_name>)  {
          pin (<input pin1>) {
               direction: input;
               …….
          }
          pin (<input pin2>) {
               direction: input;
               …….
          }
          ……<other input pins>…….
          pin (<output pin1>) {
               direction: output;
               function: <function of this pin in terms of
input pins>
               …….
          }
          pin (<output pin2>) {
               direction: output;
               function: < function of this pin in terms of
input pins>
               …….
          }
          ……<other output pins>…….
     }
}
```

In this way, a new set of libraries could be generated by modifying the input/output pin list or redefining the functionality of output pins. After the required library was generated for each experiment, a flow was developed to convert the library to the .db format compatible with the Design Compiler tool of Synopsys. This flow was developed and tested on a sample test, after which the following set of experiments were conducted.

1.2.1 Modified Inverter Cell 'INVX1'

In this experiment, one of the inverters in the library was replaced by a new standard cell which had two inverters in parallel. This experiment helped understand if multiple independent circuits in a standard cell could be understood by the Synthesis tool.

```
library (modified_lib) {
     cell (INVX1)  {
          pin (A) {
               direction: input;
               …….
          }
          pin (B) {
               direction: input;
               …….
          }
          pin (Y0) {
               direction: output;
               function: ~A
               …….
          }
          pin (Y1) {
               direction: output;
               function: ~B
               …….
          }
     }
}
```

When this modified library was used to synthesize a given RTL, it was found that Synthesis did not consider this as an inverter, and picked other inverters to synthesize the design.

1.2.2 Modified Inverter Cell with No Other Inversion Cells

In this experiment, one of the inverters in the standard cell was replaced by a new standard cell which had two parallel inverters, similar to the above experiment. Also, all the other inverters and inversion cells were removed from the library to ensure that no other inverters were picked. When Synthesis was run, it reported an error that no inverter was found in the library.

Based on the above two experiments, it can be inferred that Synthesis was not able to understand that an inverter could be located inside a cell which had 2 input pins.

1.2.3 Convert a 2-Input Cell to an Inverter

In the above experiments, a new input pin was added to the existing 1-input pin cell and it was found that the cell was not usable. A possible reason could be that adding a second input pin made the cell unusable. This was further tested in an experiment where an existing 2-input pin cell was taken and its functionality changed. In this experiment, the 2-input cell (DEC24x1) with input pins A0 and A1 was modified. The modified functionality of one of the output pins was Y0 =~A0.

```
library (modified_lib) {
     cell (DEC24X1)  {
          pin (A0) {
          direction: input;
          …….
          }
          pin (A1) {
          direction: input;
          …….
          }
          pin (Y0) {
          direction: output;
          function: ~A0
          …….
          }
          pin(Y1) {
          direction: output;
          function: A0*(~A1)
          …….
          }
          ……..
     }
}
```

When Synthesis was run, it did not detect this modified cell as an inverter. It can be inferred that a 2-input pin cell cannot be converted to an inverter by simply modifying the functionality of an output pin.

1.2.4 Convert a 2-Input Cell to an Inverter with Single Input Pin

From the above experiments, it can be concluded that the modification of the functionality of an output pin did not work. This may be because of the ‘don’t care’ input pins making the cell unusable. In this experiment, the 2-input cell (DEC24x1) with input pins A0 and A1, was modified by removing the second input pin (A1). Also, the functionality of one of the output pins was modified to Y0 =~A0. When Synthesis was run, it detected this modified cell as an inverter.

```
library (modified_lib) {
     cell (DEC24X1)  {
          pin (A0) {
               direction: input;
               …….
          }
          pin (A1) {
               direction: input;
               …….
          }
          pin (Y0) {
               direction: output;
               function: ~A0
               …….
          }
          pin (Y1) {
               direction: output;
               function: A0
               …….
          }
          ……..
     }
}
```

Based on the above observation it can be concluded that modifying the original 2-input cell to a one input pin cell made it a possible candidate in Synthesis's 'inverter search algorithm'. Hence, in general, removing pins from an existing cell works if the final number of input pins in the modified cell is equal to the number of variables in the functionality of the output pin.

1.2.5 Modify the 2-Input Cell with Different Functionality

In this experiment, a new pin, A2, was added to the modified 2-input cell from the previous experiment. The functionality of one of the output pins was also defined as Y0 = A2*A0. In order to ensure that Synthesis does not select any other AND gate, all other AND gates were removed from the library. When Synthesis was run, it detected this

modified cell as an AND gate. Based on this experiment, it can be concluded that replacing an existing input pin with a new input pin in the cell and realizing the cell's output functionality in terms of this newly added input pin gives the desired result. In general, Synthesis would consider the modified cell in its algorithm only when the number of input pins is not more than the number of independent variables in the function which it is trying to realize.

1.2.6 Generating A Complex Cell

It was found from the above experiments that adding and removing pins from a cell and modifying the functionality of its output pin in terms of new pins works for simple cells like an inverter. Since the feedstock would have complex cells, it is important to validate if these modifications would also work for complex cells. In this experiment, a 4-input cell (NAND4X0) was taken and the functionality of one of the output pins was defined as Y0 = (A4'*A3'*A2'*A1') + (A4*A3*A2*A1). Simultaneously, a similar RTL was given to the Synthesis tool, y=((~i1&~i2&~i3&~i4) | (i1&i2&i3&i4))

When Synthesis was run, it successfully detected this modified cell and synthesized the RTL using this modified cell.

```
library (modified_lib) {
     cell (NAND4X0)  {
          pin (A0) {
               direction: input;
               …….
          }
          pin (A1) {
               direction: input;
               …….
          }
          ……<other input pins >…….
          pin (Y0) {
               direction: output;
               function: (A4'*A3'*A2'*A1')+(A4*A3*A2*A1)
               …….
          }
          …….<other output pins >…...
     }
}
```

1.2.7 Modifying the Complex Cell

The complex cell generated in the above experiment was further modified by adding an extra dummy input pin to check if this extra dummy pin would cause any issues in this complex cell. The same RTL was given to the Synthesis tool, y=((~i1&~i2&~i3&~i4) | (i1&i2&i3&i4)). When Synthesis was run, this modified complex cell was not used to synthesize the RTL.

1.2.8 Conclusion

Based on the above experiments, it can be concluded that the Synthesis tool first counts the total number of input variables in the logic it is trying to synthesize. Its 'cell picking' algorithm then prunes the search space by picking only those cells that do not have more input pins than the number of independent variables in the logic.

1.3 USING .LIB FEATURES TO DEFINE FEEDSTOCK AS A CELL

The previous experiments revealed that the Synthesis tool counts the total number of input pins of the cell being considered in its 'cell picking' algorithm. Hence, some features of the .lib file were tested to accommodate this feature of the Synthesis tool and to meet the project's requirement.

1.3.1 INOUT Pins Experiment

An experiment was conducted by defining the direction of the pin while defining a gate in the library. Since a pin can have one of three possible directions- input, output or inout- defining the direction of the spare pins in a feedstock cell as inout, can help check if these pins are not counted as input pins. Thus, Synthesis was run with an RTL such that it uses that feedstock and it was found that the inout pins were still getting counted as input pins. It can be concluded that the direction of a pin as inout is not helpful.

1.3.2 TIE-HIGH and TIE-LOW Cells Experiment

After the above experiment, a second option was to make use of tie-high and tie-low cells to connect to the spare pins while doing synthesis. These cells were thus, added to the library and the dont_use option was set to false. Synthesis was then run to check if the spare pins of the feedstock would be connected with these tie-high and tie-low cells. The results indicated that the tool did not use the feedstock and synthesized the design using other fundamental gates. If the spare pins were removed from the feedstocks, then the Synthesis tool was able to use them. It can be concluded that the feedstock is not usable even after tying its spare pins to the tie cells.

1.3.3 Synthetic Libraries

While defining the library, an advanced feature allowed the user to define complex cells. This feature is particularly useful when a module is repeated several times in the design, as it has a provision to include that design in the library itself as a standard cell. But, the Verilog model of the module had to be defined and 'alib' [17] for the module had to be generated for this feature to work. Then, during synthesis, if the alib library path had been defined, the synthesizer would go through the alib library along with the target and link libraries to synthesize the design. This procedure for developing a synthetic library and alib models was used for the feedstocks. However, it was observed that the spare input pins in the feedstock prevented the synthetic models of the feedstocks from getting used by the synthesizer in its mapping algorithm.

1.4 GENERATING LIBRARIES FOR HIERARCHICAL CELLS

One important requirement for running a feedstock-based synthesis flow is characterizing each feedstock for timing, power, leakage, and other models. Since feedstocks were being built using standard cells, the effort of re-characterizing the feedstocks can be saved. However, a new flow which would take the Verilog model of the feedstock standard cells' libs and generate the hierarchical libs of the feedstock library, needed to be developed. Timing and power analysis was required to be done for feedstock with complex logic. For such feedstock, the flow was developed using the Primetime and PTPX tools from Synopsys. Once generated, the .libs could be converted to .db using the flow developed in previous experiments. Some components that were generated in the .lib were cell area, cell leakage, and clock model. For the input pins, the setup and hold time model was included. For the output pins, the fall/rise delay and fall/rise cell transition timing models were included.

Since there was no complex logic in the feedstock in the final approach. This developed flow would be useful when complex logic is added to the feedstock and timing and power analyses of different paths of the feedstock are needed to generate the .libs

1.5 DEFINING EACH FEEDSTOCK AS A LIBRARY

From the previous experiments, it is evident that defining each feedstock as a cell did not work because the Synthesis tool's 'cell picking' algorithm considers only those cells that do not have more input pins than the number of independent variables in the logic. However, since the requirement of the project is to have multiple cells in a single feedstock, a few different approaches were developed.

In one approach, each individual feedstock was defined as a unique library with the cells in each library representing the cells present in that feedstock. Whenever a feedstock was picked, the corresponding library would be active and cells from that library would be used to synthesize the design. However, it had to be ensured that the cells could only be used as many times as their instances in the library. For example, if a feedstock A had 5 inverters, then the inverter cell could be used a maximum of 5 times when the library was called. However, the standard algorithm allowed a given cell from the called library to be used any number of times. This meant that a different approach had to be adopted to meet this requirement.

In order to ensure that none of the cells were instantiated more than their count on the feedstock, a possible approach was to set the usage of each cell in the given library to a constant. However, after going through literature and library documentation, it was found that there was no such feature. An alternate solution was to uniquely define each cell present on a feedstock, in its corresponding library. However, the problem still remained as the software allows only two knobs- dont_use, and preferred [16]. Therefore,

in order to exhaustively use the cells from the library, a scanning based approach was developed, by another project member [12], wherein windows were created in the standard cell-based design and for each window, the best-fit feedstock was assigned depending on the physical location of cells. The base layer was fixed and the feedstock level netlist was generated after all the windows were mapped to different feedstocks. Finally, Conformal ECO flow was run to achieve the desired functionality.

Chapter 2: ECO and Conformal ECO

In a standard ASIC design flow, the Engineering Change Order (ECO) stage is used when there are design changes in RTL that need to be incorporated after implementation of the SOC is done. ECO does logical equivalence of the original design's netlist with the modified netlist and captures the changes in the design. It then generates the patch file which represents all these changes. This patch file is synthesized using the spare cells already present in the design which precludes the physical design of the whole SOC from scratch. After the design changes are mapped to the spare cells, the synthesized cells are swapped with their corresponding spare cells. Finally, the incremental routing of theses spare cells is done.

In the M2A2 design flow, the "exhaustive usage" feature of the ECO was used to meet the design functionality. ECO has the ability to realize the changes in the design by using only the spare cells. This feature was exploited in this project. The feedstock-based netlist generated from the scanner script was given to ECO as the golden design. Since there was no connectivity in the netlist, all the cells in this feedstock-based netlist were defined as spare cells, allowing the ECO to synthesize the whole design using these spare cells. Further optimizations were performed to meet performance once the design functionality was met.

2.1 ECO - FEATURES AND LIMITATIONS

Given the "exhaustive mapping" feature of the ECO described above, preliminary experiments were performed to confirm optimal synthesis, physically aware cell utilization, gate sizing and V_T cell swapping for higher performance design. Experiments were also conducted to validate if the ECO could synthesize large designs by using the

spare cells without being overwhelmed. These experiments and the inferences drawn from them are discussed in the following section.

2.2 EXPERIMENTS WITH ECO

The primary inputs to run ECO are the golden netlist and the revised netlist. The golden netlist has the original design and the revised netlist has the updated design which needs to be implemented. For these experiments, the first step involved the physical implementation of the golden netlist. Based on the netlist, a floorplan was generated for each experiment followed by a 'place and route' of the golden netlist. After being physical implemented, the golden netlist was saved in .dat format to convey the physical locations of the cells to ECO. Once the .dat and revised netlist were generated, ECO was run using ecoDesign [18] in the PostMask mode. ECO would re-route the golden design and try to achieve the same functionality as the revised netlist. After developing and testing this flow on a sample test, the following set of experiments were conducted to do an in-depth analysis of ECO.

2.2.1 Cell Swapping

This experiment was performed to verify the ECO flow and capture the requirements for cell swapping. The golden netlist given to the ECO had NAND and AND gates with {a, b} and {c, d} pins, respectively. The revised netlist had NAND and AND gates with swapped pins, i.e., {c, d} and {a, b} pins, respectively. Upon running ECO, it was found that cell swapping was achieved by just re-routing the design without changing the transistor layer.

2.2.2 Modifying Design's Functionality Using ECO

The goal of this experiment was to implement a MUX. In the golden netlist (Figure 8), all gates required to implement a MUX were present, but they were intentionally connected incorrectly. The revised netlist had the correct functionality of the MUX. ECO was able to successfully achieve the correct functionality (Figure 9) by simply re-routing the golden design. It was also observed that this process worked only when the instance name used in the revised netlist is the same as the golden netlist.

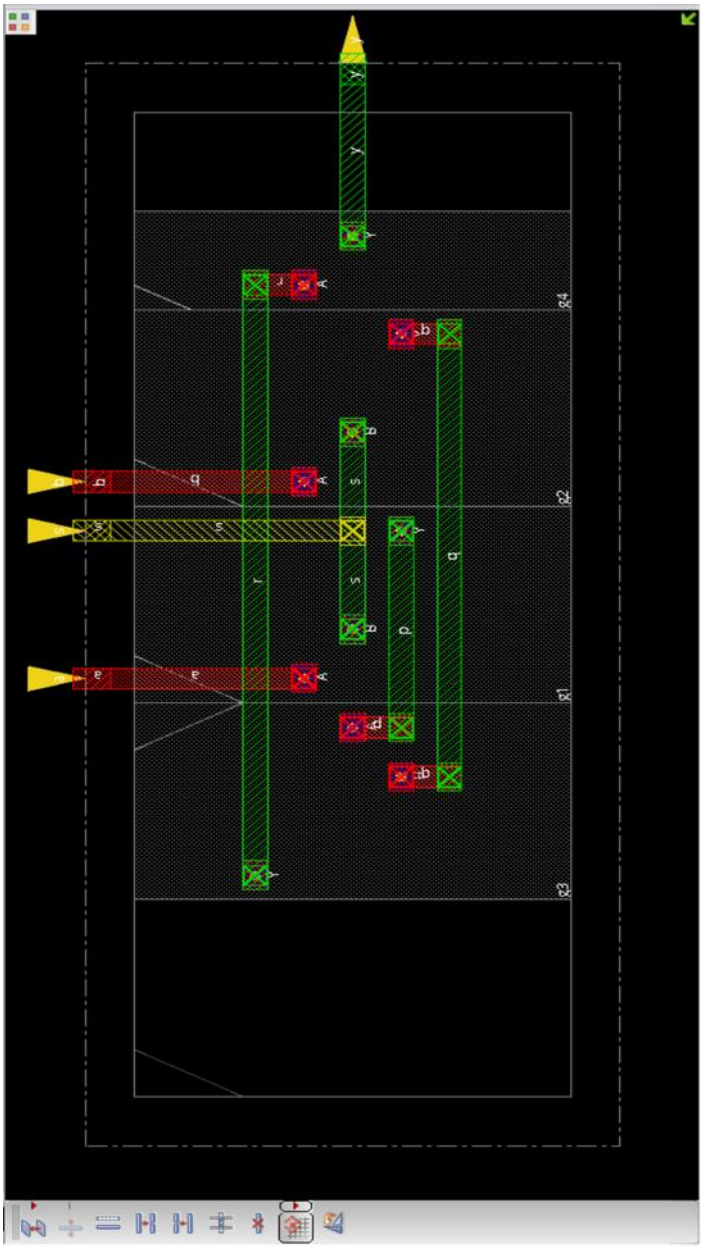


Figure 8: Initial design before running ECO

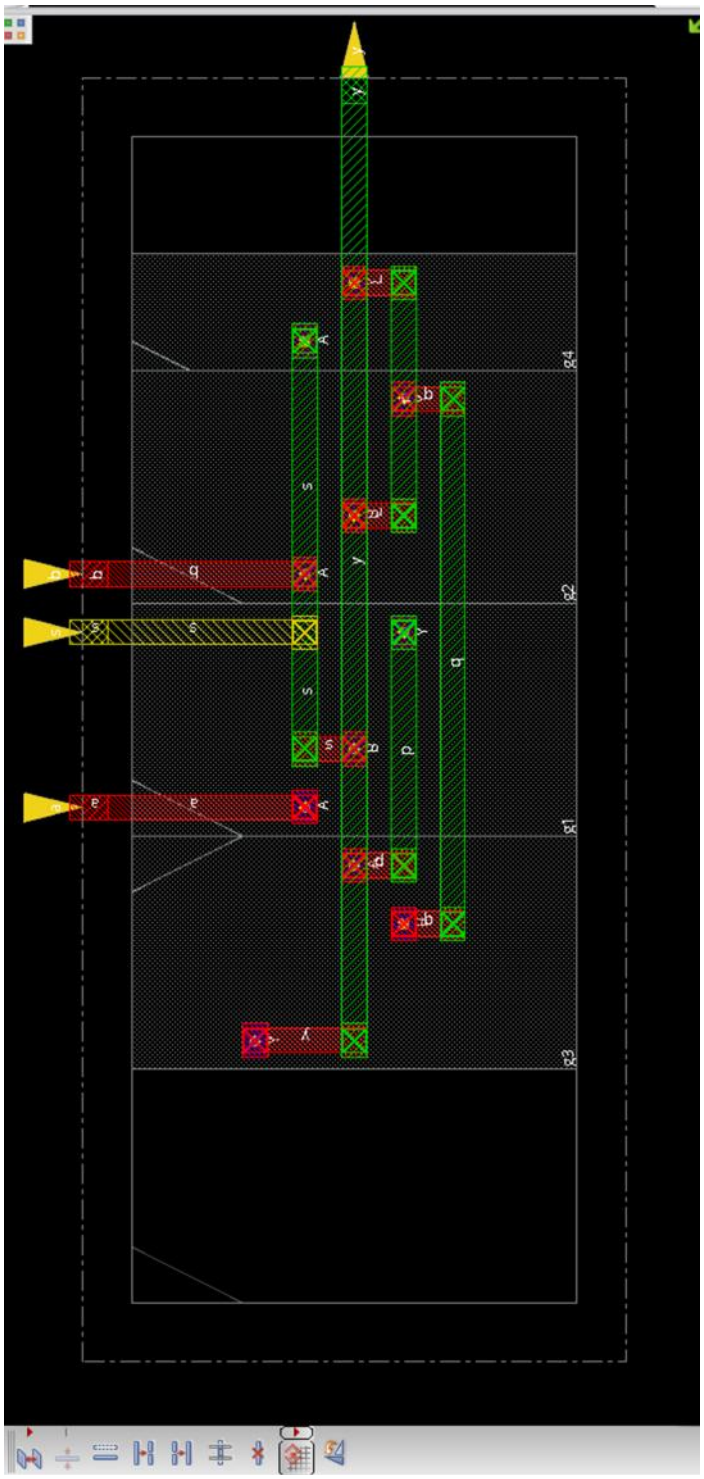


Figure 9: Revised design implemented by just re-routing

2.2.3 Replacing Gate with Another Gate with Same Instance Name

In the previous experiment, it was found that the instance name should remain the same between the golden and revised netlists. The goal of this experiment was to further analyze how ECO would work if the instance name was kept the same but its functionality was changed. The experiment was conducted by naming an AND gate and an OR gate instance in the golden and revised netlists, respectively, as G1. This resulted in an error that no spare OR gate was found in the design, when the ECO was run. This was further analyzed in another experiment when a spare OR gate was added to the golden netlist as shown below.

```
//golden netlist
module TRIAL_CHIP(….);
     AND2X1 G1 …..
     OR2X1  G2 …..
endmodule
```

```
//revised netlist
module TRIAL_CHIP(….);
     OR2X1 G1 …..
endmodule
```

Realizing that an OR gate with instance name G1 should be there in the final design, ECO checked gate G1 in the golden netlist for an OR gate, but found it to be an AND gate. It then took a spare OR gate and named that as G1 while renaming the AND gate as G1_SPARE. This '_SPARE' suffix is a default prefix which is added to all the freed cells. The user has the ability to provide a different suffix as well. The final generated netlist was as following.

```
//final ECO generated
netlist
module TRIAL_CHIP (….);
     OR2X1  G1 …..
     AND2X1 G1_SPARE…
endmodule
```

2.2.4 Implementing Full Design Using Unconnected Spare Cells

The goal of this experiment was to implement the complete functionality of the design with ECO only. In the golden netlist, enough spare gates were given to implement a MUX, but without any connection between the spare gates. In the revised netlist, a MUX was given, with completely different instance names from the golden netlist.

```
//golden netlist
module TRIAL_CHIP (a,b,sel,y);
     AND2X1 G1 …..
     AND2X1 G2 …..
     AND2X2 G3 …..
     AND2X2 G4 …..
     OR2X1  G5 …..
     OR2X2  G6 …..
     INVX1  G7 .….
     INVX2  G8 …..
endmodule
```

```
//revised netlist
module TRIAL_CHIP (a,b,sel,y);
     AND2X1 M1 (a, sel, a1)
     INVX1  M2 (sel, sel_bar)
     AND2X1 M3 (b, sel_bar, b1)
     OR2X1  M4 (a1, b1, y)
endmodule
```

As expected, ECO exited with the error – 'Design must be placed before running ECO', because M1, M2, M3, and M4 were not present in the golden netlist.

Another experiment was run to overcome this error by keeping the same instance names between the golden and revised netlists. However, the logic functionalities for the same instance name could be different between the golden and revised netlists, as shown below.

```
//golden netlist
module TRIAL_CHIP (a,b,sel,y);
     AND2X1 G1 …..
     AND2X1 G2 …..
     OR2X1  G3 …..
     INVX1  G4 .….
endmodule
```

```
//revised netlist
module TRIAL_CHIP (a,b,sel,y);
     AND2X1 G1 (a, sel, a1)
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y)
endmodule
```

When ECO was run, it successfully implemented the same functionality as the revised netlist by only re-routing the design without changing the base layer. The final generated netlist was as following.

```
// final ECO generated netlist
module TRIAL_CHIP (a, b, sel, y);
     AND2X1 G1 (a, sel, a1);
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y)
endmodule
```

2.2.5 Implementing Modified Design

In the previous experiment, it was found that ECO was able to implement the full design using unconnected spare cells. Another experiment was designed to verify cases where there might be some functionality in the initial golden netlist but which needs to be changed to a completely different design. The capability of ECO in this scenario was tested by providing an already routed design as the golden netlist, and another design in the revised netlist, as shown below.

```
//golden netlist
module TRIAL_CHIP (a,b,sel,y);
     INVX1  G1 (a, p);
     INVX1  G2 (b, q);
     INVX1  G3 (sel, r);
     INVX1  G4 (sel, y);
     AND2X1 G5  …..
     AND2X1 G6  …..
     AND2X2 G7  …..
     AND2X2 G8  …..
     OR2X1  G9  …..
     OR2X2  G10 …..
     INVX1  G11 .….
     INVX2  G12 …..
endmodule
```

```
// revised netlist
module TRIAL_CHIP (a,b,sel,y);
     AND2X1 G1 (a, sel, a1);
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y)
endmodule
```

In this experiment, ECO was able to implement the revised design using the spare and freed cells from the golden netlist. The generated netlist is shown below with the gates that were finally used to implement the design, shown in purple. The gates that have _SPARE as the suffix are the freed cells.

```
// final ECO generated netlist
module TRIAL_CHIP (a, b, sel, y);
     INVX1  G1_SPARE …..
     INVX1  G2_SPARE …..
     INVX1  G3_SPARE …..
     INVX1  G2 (sel, sel_bar)
     AND2X1 G1 (a, sel, a1);
     AND2X1 G3 (b, sel_bar, b1)
     AND2X2 G7_SPARE …..
     AND2X2 G8_SPARE …..
     OR2X1  G4 (a1, b1, y)
     OR2X2  G10_SPARE …..
     INVX1  G11_SPARE …..
     INVX2  G12_SPARE …..
endmodule
```

It can be inferred from this experiment that each cell was taken from the revised netlist one by one, by ECO. If ECO was unable to locate an instance with the same name in the golden netlist as the picked cell, it exited with an error that design must be placed before running ECO. If it was able to find a gate with the same instance name, then it would check its functionality, and use the cell in the final design only if the functionality matched with that that of the cell in the revised netlist. If the functionalities did not match, then ECO would replace the cell with an available spare cell of the desired functionality and add a suffix (_SPARE) to the name of this freed cell. Finally, after doing this process for all the cells in the design, ECO would re-route the design without changing the base layer.

2.2.6 Physically Aware Design Optimization

From the above experiments, it was also concluded that ECO had the capability to implement the functionality of the full design even if there was no connectivity in the initial design. The next step was to investigate the optimality of the design implementation by ECO. For example, for optimal implementation, ECO should select the closest spare cell, among all spare cells of the same type. This capability was tested by giving the same golden and revised netlists, as in the previous experiment. One AND gate was added to the golden netlist to check if ECO picked the closer AND gate or not.

Based on the generated re-routed design, it was found that ECO picked the closer AND gate and did not use the AND gate located farther away. Hence, it could be concluded that ECO was able to achieve physically aware design optimization.

2.2.7 Gate-Size and V_T Aware Design Optimization

The goal of this experiment was to find whether ECO could do gate sizing or swapping of cells with different V_T, within the available spare cells. In this experiment, several spare cells with different drive strength were specified as the golden netlist. In the revised design, an un-optimized MUX design was given.

```
//golden netlist
module TRIAL_CHIP(a,b,sel,y);
     INVX1  G1 …..
     INVX1  G2 …..
     INVX1  G3 …..
     INVX1  G4 …..
     AND2X1 G5  …..
     AND2X1 G6  …..
     AND2X2 G7  …..
     AND2X2 G8  …..
     OR2X1  G9  …..
     OR2X2  G10 …..
     INVX1  G11 .….
     INVX2  G12 …..
endmodule
```

```
// revised netlist
module TRIAL_CHIP (a, b, sel, y);
     AND2X1 G1 (a, sel, a1);
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y)
endmodule
```

ECO successfully routed the design based on the revised netlist. It also optimized the design by picking closer cells, but did not optimize the timing path by picking cells of different drive strength. The generated netlist was as following.

```
// final ECO generated netlist
module TRIAL_CHIP (a, b, sel, y);
     INVX1  G1_SPARE …..
     INVX1  G2_SPARE …..
     INVX1  G3_SPARE …..
     INVX1  G2 (sel, sel_bar)
     AND2X1 G1 (a, sel, a1);
     AND2X1 G3 (b, sel_bar, b1)
     AND2X2 G7_SPARE …..
     AND2X2 G8_SPARE …..
     OR2X1  G4 (a1, b1, y)
     OR2X2  G10_SPARE …..
     INVX1  G11_SPARE …..
     INVX2  G12_SPARE …..
endmodule
```

Even though spare cells of different drive strengths were available to the ECO, the cell types in the final generated netlist and the revised netlist were the same. Hence, it can be concluded from above experiment that ECO did not change the flavor of the cell and treated the cell's drive strength as golden.

2.2.8 Conclusion

The above experiments reveal that the ECO has two requirements. The first requirement is that the gates needed in the new netlist should be present in the golden netlist as well. The gates could either be spare cells or used in the golden design. The second requirement is that the names of the instances in the new netlist must match the name of any one instance in the golden netlist, irrespective of the functionality or physical location of that instance. After meeting these requirements, ECO uses the spare cells and does re-routing to achieve the correct functionality.

From the optimization point of view, ECO successfully achieved physically aware optimization and picked the spare cells that were closer. However, it did not optimize the

timing path by gate sizing the cells or swapping the spare cells. It did not consider or optimize the drive strength of the cells and treated it as golden.

2.3 CONFORMAL ECO

Based on the previous experiments with ECO, it was observed that ECO is a good candidate to implement the full functionality of the design. Its ability to optimize physical locations while picking the cells is a useful feature for this project. At the same time, ECO had a limitation that it would not optimize the design by gate sizing or swapping different V_T cells. This feature is extremely important for this project to enable the optimization of a high performance or low power design.

In order to overcome this limitation, a new flow was considered with Conformal ECO [19]. The Conformal ECO flow was setup by loading the golden and revised designs, that had already been flattened and uniquified. The option to add pin constraints was also added to the flow, to allow the constraint of a constant value for scan chains. Next, the logical equivalence of the golden and revised netlist was conducted at all the comparison points and a patch file was generated with all the differences. This patch file would be used to modify the golden netlist into the revised netlist.

After the patch file was generated, the list of spare cells was also provided. In the absence of any functionality in the golden design, all the cells were spare and indicated with the wildcard '*'. Along with the spare cell list, the unique physical location of each spare cell was also given through a 'def' file. This would allow the tool to optimize the design based on the physical location of each individual spare cell. After both, the patch file and spare cells list had been provided, the patch file was optimized based on the available spare cells and their physical coordinates. The constraint file was also provided

in this step to allow the tool to know the timing constraints of the design. Conformal ECO uses the RTL Compiler (RC) to optimize the design. After doing the optimization, Conformal ECO generated the spare cell mapping file (swapcells.tcl) which mapped the spare cells to those cells that were used to implement the revised design. This spare cell mapping file would later be needed in the Encounter Digital Implementation (EDI) flow in order to re-route the design and conduct the physical design of the generated netlist.

2.3.1 Gate-Size and V_T Aware Design Optimization

The flow described above was run to see if Conformal ECO would be able to optimize the design by doing gate sizing. The golden netlist that was provided to the Conformal tool had a number of spare cells with different drive strengths. The revised netlist had a design of a MUX, which could be further optimized by gate sizing. The timing report and timing slack of the critical path, before and after running Conformal ECO based optimization, was as following.

```
================================================================

       Pin             Type      Fanout Load Slew Delay Arrival
                                        (fF) (ps)  (ps)    (ps)
----------------------------------------------------------------
(clock vclk1)         launch                                 0 R
(dtmf.sdc_line_19)    ext delay                 +5000     5000 R
s                     in port         3  9.0    0    +0     5000 R
cfm_eco_patch_1/s
  m_4_SPARE/A                                          +0     5000
  m_4_SPARE/Y         INVX1           1  2.0   18   +19     5019 F
  m_2_SPARE/B                                          +0     5019
  m_2_SPARE/Y         AND2X1          1  2.7   49  +120     5139 F
  m_3_SPARE/B                                          +0     5139
  m_3_SPARE/Y         OR2X1           1  2.0   50  +148     5286 F
  m_5_SPARE/D         DFFX1                                +0     5286
  m_5_SPARE/CK        setup                         0  +162     5448 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock vclk1)         capture                                5500 R
----------------------------------------------------------------
Cost Group   : 'vclk1' (path_group 'vclk1')
Timing slack :      52ps
Start-point  : s
End-point    : cfm_eco_patch_1/m_5_SPARE/D
```

Figure 10: Timing Report – Before Running Conformal ECO Optimizations

```
============================================================

       Pin              Type       Fanout Load Slew Delay Arrival
                                          (fF) (ps)  (ps)    (ps)
------------------------------------------------------------------
(clock vclk1)         launch                                    0 R
(dtmf.sdc_line_19)    ext delay                          +5000  5000 R
s                     in port          3 10.3    0      +0     5000 R
cfm_eco_patch_1/s
  g34/A                                                   +0     5000
  g34/Y               INVX1            1  3.3   23     +22     5022 F
  g33/A                                                   +0     5022
  g33/Y               AND2X2           1  3.8   38     +86     5108 F
  g31/B                                                   +0     5108
  g31/Y               OR2X2            1  2.0   42    +130     5238 F
  m_5_SPARE/D         DFFX1                             +0     5238
  m_5_SPARE/CK        setup                0  +160    5398 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock vclk1)         capture                                5500 R
------------------------------------------------------------------
Cost Group   : 'vclk1' (path_group 'vclk1')
Timing slack :     102ps
Start-point  : s
End-point    : cfm_eco_patch_1/m_5_SPARE/D
```

Figure 11: Timing Report – After Running Conformal ECO Optimizations

It can be seen in Figure 10 and 11 that Conformal ECO was able to optimize the design if spare cells of different drive strengths were present in the golden netlist. The initial timing slack of the MUX design, which was given as an input to the Conformal ECO, was 52 ps. By using the spare AND2X2 and OR2X2 cells present in the design, the timing slack was improved to 102 ps. These results confirmed that Conformal ECO is a good candidate to meet the M2A2 design flow requirements.

2.3.2 Feedstock-Based Design Implementation

The aim of the M2A2 design flow is to implement the design using the Feedstock library. A set of experiments were run to verify the compatibility of the flow described in the above section with the feedstock-based approach of this project. There are two ways in which the feedstock can be defined in the golden netlist: flattened or hierarchical. The

following experiments were run for both types of feedstock definition to verify their compatibility with this project's requirements.

2.3.3 Hierarchical Feedstock

The goal of this experiment was to define each feedstock hierarchically along with multiple instantiations in the top module, as per the requirement of the design. In the golden netlist, four feedstocks namely feedstock1, feedstock2, feedstock3 and feedstock4 were defined with spare gates inside them. Each feedstock was instantiated only once in the top module. The revised netlist was given a sub-optimal design of a MUX. The experiment was run to verify whether the Conformal ECO had the ability to understand this feedstock-based golden netlist and implement the design optimally. The golden and revised netlist were as follows.

```
//golden netlist
module feedstock1
     AND2X1 …..
     AND2X1 …..
     AND2X2 …..
     AND2X2 …..
endmodule

module feedstock2
     OR2X1 …..
     OR2X1 …..
     OR2X2 …..
     OR2X2 …..
endmodule

module feedstock3
     INVX1 …..
     INVX1 …..
     INVX2 …..
     INVX2 …..
endmodule

module feedstook4
     DFFX1 …..
     DFFX1 …..
endmodule

module TRIAL_CHIP (a,b,
sel,clk,y);
     feedstock1 f1 …..
     feedstock2 f2 …..
     feedstock3 f3 …..
     feedstock4 f4 …..
endmodule
```

```
// revised netlist
module TRIAL_CHIP (a,b,sel,clk,y);
     AND2X1 G1 (a, sel, a1)
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y1)
     DFFX1  G5 (y1, clk, y);
endmodule
```

A ‘def’ of the golden netlist was also generated to get the physical information of each feedstock with its cells. Conformal ECO was run with the golden netlist, its ‘def’, and the revised netlist. It was found that Conformal ECO was able to understand this

feedstock-based golden netlist, implement the revised netlist and also optimize the design using spare cells by gate-sizing. The final generated netlist was as following.

```
//// final Conformal ECO generated netlist
module feedstock1
     AND2X1 …..
     AND2X1 …..
endmodule

module feedstock2
     OR2X1 …..
     OR2X1 …..
     OR2X2 …..
endmodule

module feedstock3
     INVX1 …..
     INVX2 …..
     INVX2 …..
endmodule

module feedstook4
     DFFX1 …..
endmodule

module TRIAL_CHIP (a, b, sel, clk, y);
     feedstock1 f1 …..
     feedstock2 f2 …..
     feedstock3 f3 …..
     feedstock4 f4 …..
     AND2X2 ECO2inst_1 (a, sel, a1);
     INVX1  ECO2inst_2 (sel, sel_bar);
     AND2X2 ECO2inst_3 (b, sel_bar, b1);
     OR2X2  ECO2inst_4 (a1, b1, y1);
     DFFX1  ECO2reg_G5 (y1, clk, y);
endmodule
```

In the final revised netlist, it was observed that the spare cells that were used to implement the design, were removed from the feedstock module itself. For instance, two AND2X2 cells were used in the top module to implement the design. These cells were also removed from the 'feedstock1' module definition, which originally contained two AND2X1 and two AND2X2 cells. Similarly, the module definitions of feedstock2, feedstock3 and feedstock4 were also modified based on the cells that were used.

In this experiment, modifying the feedstock definition did not lead to problems since each feedstock was instantiated only once. However, there could be cases where each feedstock is instantiated multiple times in the top module making this method unusable. Hence, an experiment was needed to check the ability of the tool for the above instance.

2.3.4 Hierarchical Feedstock with Multiple Instantiation

Based on an observation from the above experiment, it is possible that the Conformal tool fails if a feedstock is instantiated multiple times in the top module. The goal of this experiment is to verify this possibility. In this experiment, the golden netlist had three feedstocks, namely feedstock1, feedstock2 and feedstock3. In the top module, feedstock1 was instantiated twice and feedstock2 and feedstock3 were both instantiated once. The revised netlist was kept the same as that in the previous experiment.

```
//golden netlist
module feedstock1
     AND2X1 …..
     AND2X2 …..
     INVX1 …..
endmodule

module feedstock2
     OR2X1 …..
     OR2X1 …..
     OR2X2 …..
     OR2X2 …..
endmodule

module feedstook3
     DFFX1 …..
     DFFX1 …..
endmodule

module TRIAL_CHIP (a,b,sel,clk,
y);
     feedstock1 f1_1 …..
     feedstock1 f1_2 …..
     feedstock2 f2 …..
     feedstock3 f3 …..
endmodule
```

```
// revised netlist
module TRIAL_CHIP (a,b,sel,clk,y);
     AND2X1 G1 (a, sel, a1);
     INVX1  G2 (sel, sel_bar)
     AND2X1 G3 (b, sel_bar, b1)
     OR2X1  G4 (a1, b1, y1)
     DFFX1  G5 (y1, clk, y);
endmodule
```

As in the above experiment, a ‘def’ of the golden netlist was generated to get the physical information of each feedstock and the cells inside it. Conformal ECO was then run with the golden netlist, its ‘def’ and the revised netlist. Upon running the tool, no error was observed because of the multiple instantiation of feedstock1, and the revised netlist was successfully implemented. The final generated netlist was as given below.

```
// final Conformal ECO generated netlist

module feedstock1
     AND2X1 …..
     AND2X2 …..
     INVX1 …..
endmodule

module feedstock2
     OR2X1 …..
     OR2X1 …..
     OR2X2 …..
endmodule

module feedstook3
     DFFX1 …..
endmodule

module TRIAL_CHIP (a, b, sel, clk, y);
     feedstock1 f1_1 …..
     feedstock1 f1_2 …..
     feedstock2 f2 …..
     feedstock3 f3 …..
     AND2X2 ECO2inst_1 (a, sel, a1);      ← f1_1
     INVX1     ECO2inst_2 (sel, sel_bar); ← f1_1
     AND2X2 ECO2inst_3 (b, sel_bar, b1);  ← f1_2
     OR2X2    ECO2inst_4 (a1, b1, y1);
     DFFX1    ECO2reg_G5 (y1, clk, y);
endmodule
```

From the generated netlist, it can be observed that when a cell was used from a feedstock, which was instantiated multiple times, it was not removed from the feedstock module definition. For example, the INVX1 cell was not removed from the feedstock1 module definition. However, if a feedstock was instantiated only once, then its module definition was modified. For example, feedstock2 and feedstock3 modules were modified since they were used only once in the top module.

From this observation, it can be concluded that defining feedstock hierarchically would not work for this project's requirements. This is because some cells would be repeated in this approach leading to a change in the base layer, which was not allowed in this project. More experiments were run by modifying the 'report eco change', and 'write eco design' commands, but some cells were still being repeated. The hierarchical approach was then abandoned and the flattened feedstock-based approach was tested as described below.

2.3.5 Flattened Feedstock

The aim of this approach was to have unique feedstocks that would be flattened [12] in the top module. The following experiment was devised to achieve this goal. In the golden netlist, four feedstocks namely feedstock1, feedstock2, feedstock3 and feedstock4 were defined with spare gates inside them. In the top module, each feedstock was instantiated once, and in the revised netlist, a sub-optimal MUX design was given. The goal of this experiment was to check whether Conformal ECO would successfully understand this feedstock-based golden netlist and implement the design optimally. The golden and revised netlist were as shown in Figure 12 and 13 respectively.

```
module TRIAL_CHIP (a,b,s,clk,y) ;

input a,b,s,clk;
output y;

AND2X1 F1_U_1 (.A ( F1_IN_1 ) , .B ( F1_IN_2 ) , .Y ( F1_OUT_1 )) ;
AND2X2 F1_U_2 (.A ( F1_IN_3 ) , .B ( F1_IN_4 ) , .Y ( F1_OUT_2 )) ;
INVX1 F1_U_3 (.A ( F1_IN_5 ) , .Y ( F1_OUT_3 )) ;
AND2X1 F2_U_1 (.A ( F2_IN_1 ) , .B ( F2_IN_2 ) , .Y ( F2_OUT_1 )) ;
AND2X2 F2_U_2 (.A ( F2_IN_3 ) , .B ( F2_IN_4 ) , .Y ( F2_OUT_2 )) ;
INVX1 F2_U_3 (.A ( F2_IN_5 ) , .Y ( F2_OUT_3 )) ;
OR2X1 F3_U_1 (.A ( F3_IN_1 ) , .B ( F3_IN_2 ) , .Y ( F3_OUT_1 )) ;
OR2X1 F3_U_2 (.A ( F3_IN_3 ) , .B ( F3_IN_4 ) , .Y ( F3_OUT_2 )) ;
OR2X2 F3_U_3 (.A ( F3_IN_5 ) , .B ( F3_IN_6 ) , .Y ( F3_OUT_3 )) ;
OR2X2 F3_U_4 (.A ( F3_IN_7 ) , .B ( F3_IN_8 ) , .Y ( F3_OUT_4 )) ;
DFFX1 F4_U_1 (.D ( F4_IN_1 ) , .CK ( F4_IN_2 ) , .Q ( F4_OUT_1 ) , .QN ( F4_OUT_2 )) ;
DFFX1 F4_U_2 (.D ( F4_IN_3 ) , .CK ( F4_IN_4 ) , .Q ( F4_OUT_3 ) , .QN ( F4_OUT_4 )) ;
endmodule
```

Figure 12: Golden Netlist – Flattened Feedstock Based

```
module TRIAL_CHIP(a, b, s, y, clk);
  input a, b, s, clk;
  output y;
  wire a, b, s;
  wire clk;
  wire y;
  wire p;
  wire q;
  wire r;
  wire t;

  AND2X1 m_1(.A (a), .B (s), .Y (p));
  INVX1 m_4(.A(s),.Y(r));
  AND2X1 m_2(.A (b), .B (r), .Y (q));
  OR2X1 m_3(.A (p), .B (q), .Y (t));

  DFFX1 m_5(.D(t), .CK(clk), .Q(y), .QN());
endmodule
```

Figure 13: Revised Netlist

The golden netlist was the same as that in the experiment of section 2.3.4, but flattened. As in earlier experiments, the 'def' of the golden netlist was also generated and Conformal ECO was run. The final generated netlist is given below in Figure 14.

```
module TRIAL_CHIP (a,b,s,clk,y);
   input a,b,s,clk;
   output y;

   // Internal wires
   AND2X1 F1_U_1 (.Y(F1_OUT_1), .B(F1_IN_2), .A(F1_IN_1));
   AND2X2 F1_U_2 (.Y(F1_OUT_2), .B(F1_IN_4), .A(F1_IN_3));
   AND2X1 F2_U_1 (.Y(F2_OUT_1), .B(F2_IN_2), .A(F2_IN_1));
   OR2X1 F3_U_1 (.Y(F3_OUT_1),  .B(F3_IN_2), .A(F3_IN_1));
   OR2X1 F3_U_2 (.Y(F3_OUT_2),  .B(F3_IN_4), .A(F3_IN_3));
   DFFX1 F4_U_2 (.QN(F4_OUT_4), .Q(F4_OUT_3), .D(F4_IN_3), .CK(F4_IN_4));
   DFFX1 ECO2reg_m_5 (.QN(ECO2net_6), .Q(y), .D(ECO2net_1), .CK(clk));
   OR2X2 ECO2inst_1 (.Y(ECO2net_1), .B(ECO2net_4), .A(ECO2net_2));
   INVX1 ECO2inst_2 (.Y(ECO2net_2), .A(ECO2net_3));
   OR2X2 ECO2inst_3 (.Y(ECO2net_3), .B(ECO2net_5), .A(s));
   AND2X2 ECO2inst_4 (.Y(ECO2net_4), .B(a), .A(s));
   INVX1 ECO2inst_5 (.Y(ECO2net_5), .A(b));
endmodule
```

Figure 14: Final Conformal ECO Generated Netlist

It can be observed that there was no cell repetition and the cell count in the final netlist was the same as that in the golden netlist. Also, the spare cells that were used to implement the design were renamed with a prefix 'ECO2inst_' that could also have a user-defined format. The names of the unused cells were not modified and the unused cells were left in the final design as spare cells. Based on the results of this experiment, it can be concluded that Conformal ECO with flattened feedstock served the requirements of this project.

2.3.6 Final Conformal ECO Flow

After running all the above experiments and analyzing different features of Conformal ECO, a flow which would serve the project requirements was developed. In the first step, the customer RTL would be taken and the standard cell-based ASIC flow would be run. RTL would be synthesized to generate the gate level netlist. Then, it would be placed, along with synthesis of the clock tree and routing of all other nets. Also, a different standard optimization would be conducted to get the required timing numbers and close the design.

After the customer's RTL was physically designed using the conventional standard cell-based ASIC flow, the feedstock-based flow would start. First, the scanner script, developed as part of this project by a collaborator [12], would be run. It would divide the layout of the final ASIC standard cell-based design into windows. The script would also parse through each window and assign the best-fit feedstock from the feedstock library to each window. The output of this scanner script would thus be a feedstock based netlist with all the unconnected cells. This feedstock-based netlist would also define the base layer as these feedstocks would be physically placed based on the location of the scanning window. After this stage of the design, the base layer would not be modified.

In the next stage, Conformal ECO would be run by first defining this feedstock-based netlist as the golden netlist, where all the cells would be defined as spare cells. The revised netlist would be the standard cell-based netlist. Conformal ECO would analyze the golden and revised netlists and generate the patch file. After generation of the patch file, the list of spare cells would be provided. In this project, since there would be no functionality in the golden design and all the cells would be spare, the wildcard * would be used. Also, along with the spare cell list, their unique physical locations would also be

given through the ‘def’ of the golden netlist. This would give the Conformal ECO tool the ability to optimize the design, based on the physical location of the individual spare cells. After both patch file and spare cells list are provided, the patch would be optimized based on the available spare cells and their physical coordinates. The constraint file would also be provided in this step to allow the tool to have the knowledge of the timing constraints of the design. Conformal ECO would run the RC as its core engine to optimize the design. Upon completion of optimization, the tool would generate the spare cell mapping file (swapcells.tcl) which would map the spare cells to those cells that are used to implement the revised design. This spare cell mapping file would later be needed in the EDI flow to re-route the design and perform the physical design of the generated netlist. The Conformal ECO flow diagram is shown in Figure 15.

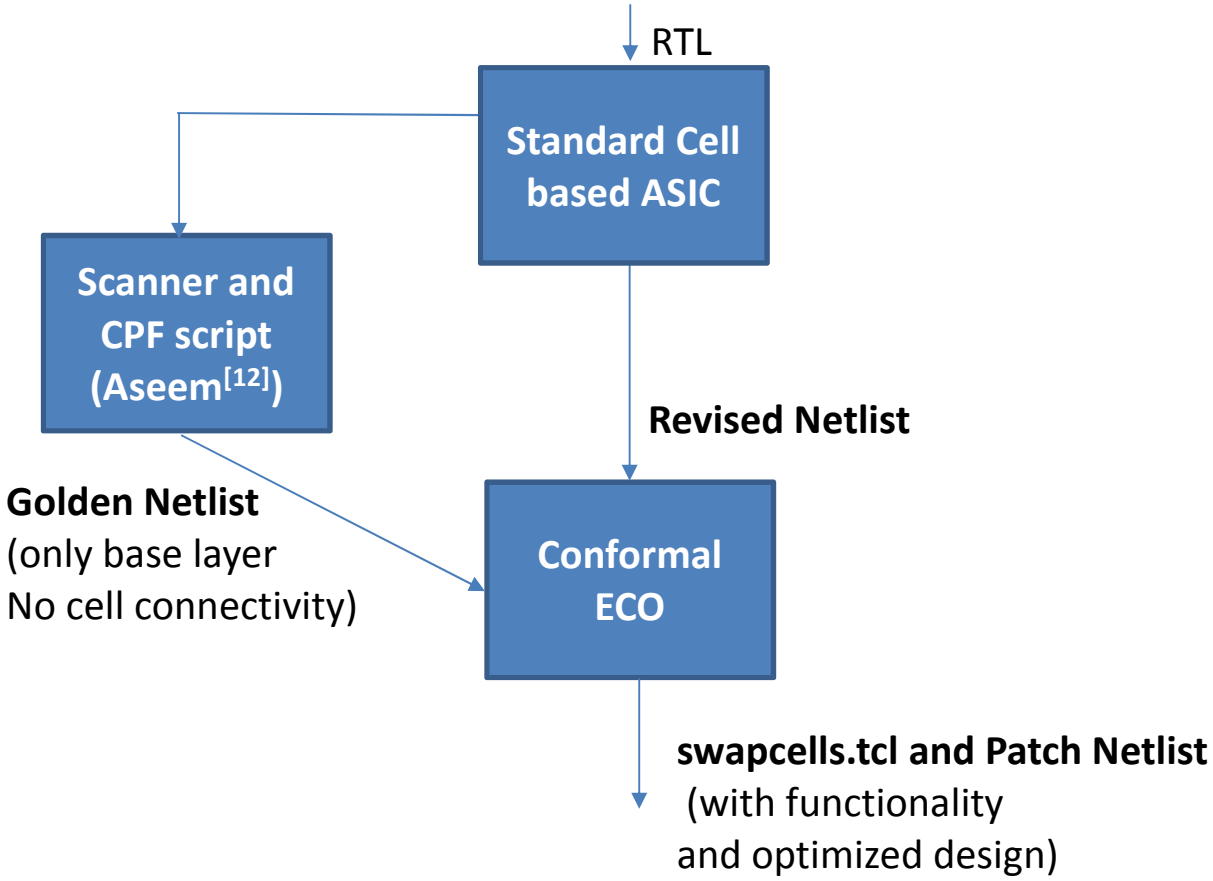


Figure 15: Conformal ECO Flow

Chapter 3: Physical Design – Post Mask EDI

In the standard ASIC design flow, the physical design stage [20] involves the physical placement of the design based on the synthesized netlist, building of the clock tree, routing of the nets and finally, fixing of the Setup/Hold violations. During this process, the design is incrementally optimized. For example, when building the clock tree, clock buffers are added to distribute the wire load and get a zero skew clock tree. These incremental changes affect the base layer.

However, in the M2A2 design flow, the base layer is already fixed with no scope of adding any new cells. Hence, the existing flow had to be analyzed and modified to meet this requirement. Different experiments were performed to first benchmark and analyze the existing flow and verify if the existing tool supports this project's "post mask" requirement. Following this analysis, the requirement to modify the flow was captured. Different experiments were conducted to obtain and validate the modified flow which would satisfy the project's requirements.

3.1 Placement (Spare Cell Mapping) – Post Mask EDI Flow

The goal of this stage in the design process was to physically place [21] the netlist generated by the Conformal ECO. As discussed earlier, the base layer was defined in the scanner script algorithm (as discussed in section 2.3.5), because all the feedstocks were already placed. In this stage, there would not be any placement of cells or changing of cell locations. However, these spare cells (present inside the feedstock) that are already placed would be mapped to the instances used in the Conformal ECO netlist. The spare cell mapping file (as discussed in section 2.3.5), generated by the Conformal ECO, would be used for this purpose.

The first stage of this process involved providing the synthesized patch netlist generated by the Conformal ECO. Then, the original ‘def’, which was also an input to the Conformal ECO, would be loaded. This def would have the physical placement of all the feedstocks, implying that it would also define the physical locations of all the spare cells present inside the feedstocks. Finally, the spare cell mapping file would be given to the flow. When the flow would be run, it would replace the spare cells with the cells present in the patch netlist generated by the Conformal ECO, based on the one-to-one mapping defined in the spare cell mapping file. This would give the physically placed netlist of the design implemented using feedstocks. The Post-Mask EDI Placement flow is shown in Figure 16.

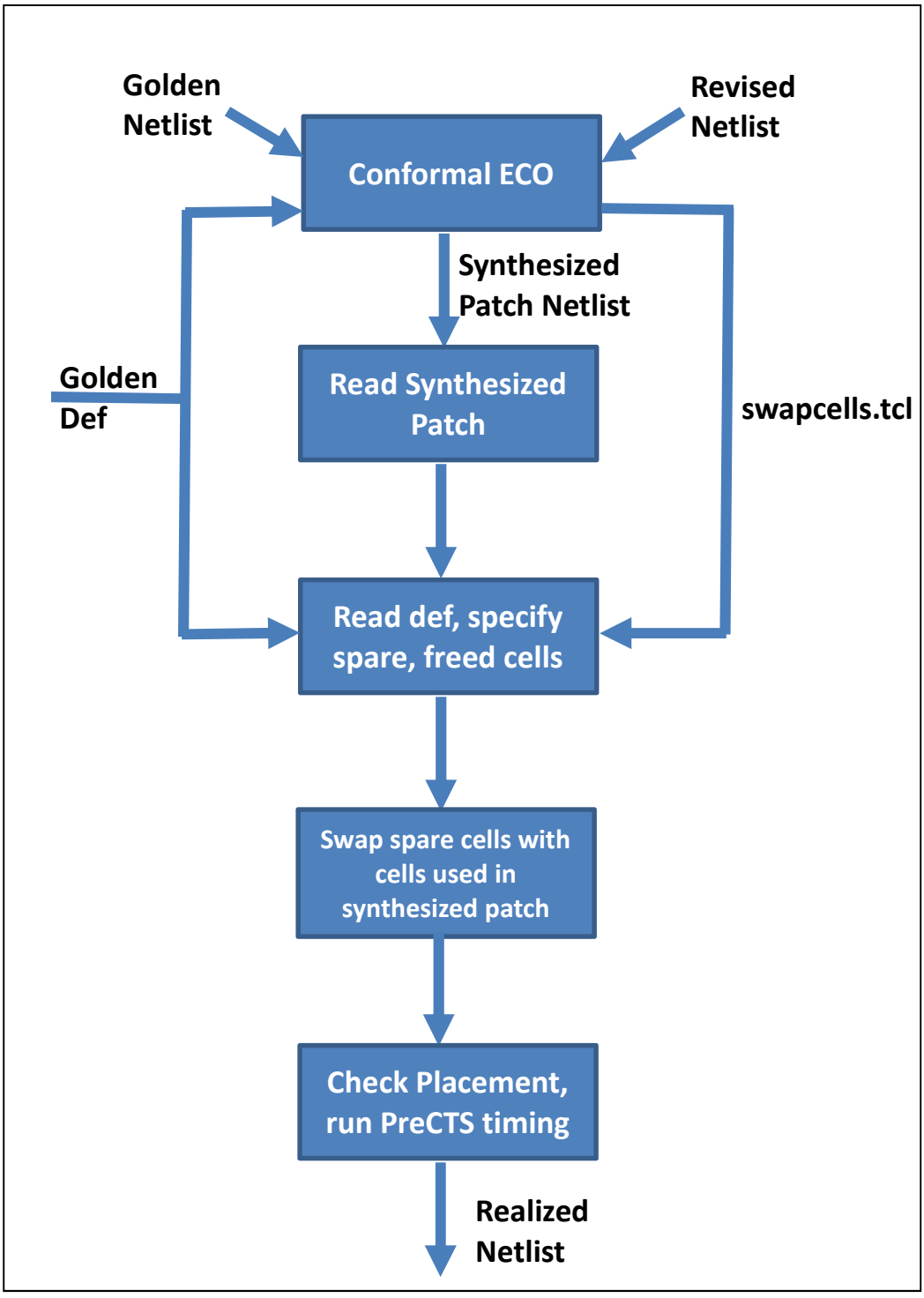


Figure 16: Post Mask EDI Placement Flow

3.2 CLOCK TREE SYNTHESIS – POST MASK

The Clock Tree Synthesis (CTS) stage of the design flow is crucial from a timing and power point of view. The clock tree is built just after the placement stage to ensure availability of routing resources. A good clock tree is mandatory for the design to meet timing requirements. Significant optimization is done to implement a zero skew clock tree with minimum slew. In the standard ASIC flow, during the CTS stage [22], it is possible to add new cells, such as clock buffers, inverters etc., to the design. There is also a feature allowing pre-placed cells to be moved to get a zero skew clock tree. Having all these knobs makes it possible to get a good clock tree with minimum skew.

In M2A2 design flow, the constraint is that the base layer cannot be modified. This constraint restricted the features that were available in existing clock tree generation tools. The only option remaining was to rely on routing to get the desired clock tree. However, dependence on routing has two drawbacks. The first drawback is that it would use substantial routing resources in order to take 'intentional' detours to achieve zero skew. Secondly, this would also increase the load and consequently, the transition time of the clock net would be very high.

Because of these issues, a new flow was needed that would enable the insertion of clock buffers to overcome these problems, without compromising the constraint of not modifying the base layer. A potential solution was inserting spare clock buffers in the clock tree as these buffers were available in the design. The challenge with this solution was that the tool would have to take detours to reach the buffer as the location of these spare buffers was fixed. Also, it had to be ensured that there were sufficient spare clock buffers in the design, failing which the tool would have to go far to find a clock buffer.

In our interactions with the industry [23], it was also found that there was no feature for implementing a full clock tree using spare cells. Since CTS is a critical step in

the design process, generation of the clock tree is done in the pre-mask stage with no existing solution for generating the clock tree in post-mask stage. This was a significant development, which motivated the design of experiments for the trial and analysis of different flows.

3.2.1 Generating Clock Tree through Routing

In this experiment, a clock tree was implemented by routing. The goal of this experiment was to check a clock tree implemented using only the routing resources. First, the clock and its source pin was defined. The extraction mode was then set to pre-route and the ‘ccopt’ based routing command was called in Innovus. The generated clock has been shown in the Figure 17.

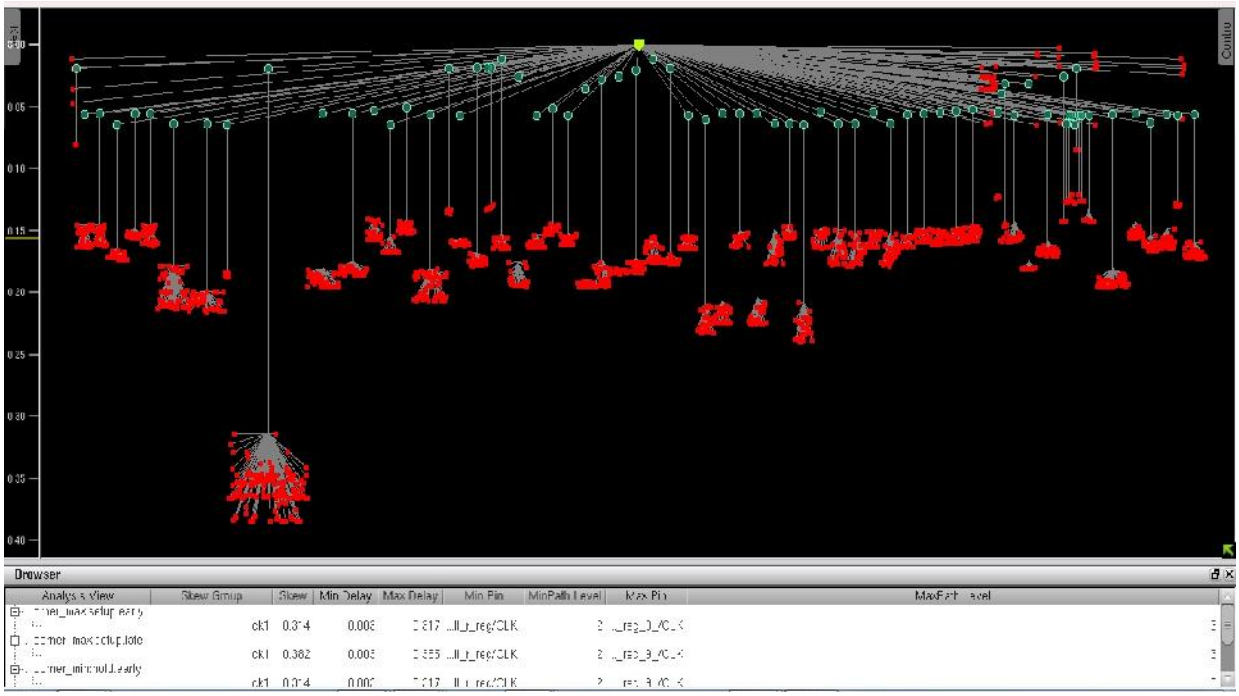


Figure 17: Generated Clock Tree using Routing

It can be seen from this example that the clock tree could be implemented using only routing. However, the clock skew could be poor, thus impacting the timing of the design. Also, the clock slew could be very high, leading to a high power loss and an increased probability of failures in the design. Additional experiments were then conducted to add buffers in the clock tree.

3.2.2 CTS with Conformal ECO

From the above experiment, it was evident that a better flow was needed to generate the clock tree instead of relying on routing only. A couple of experiments were performed to check the availability of a feature in the tool that could specify the spare clock buffer cells. Although one such feature, where the designer could specify the type of clock buffers and inverters for implementing the clock tree, was identified, these clock

buffers and inverters had to be the cells from the library and not the spare cells. This would violate 'exhaustive' usage of the spare clock buffers already present in the design.

Hence, a new flow was developed to make use of the spare clock buffers. The clock tree generation in post mask could be divided into two sub tasks. First, to insert clock buffers in the tree. Second, map these buffers to the spare cells present in the design. The first task could be done using the current tools. However, tool should be constrained in such a way that it would give optimal clock tree based on the available spare cells. Once the clock tree is built, the clock buffers used to build the clock tree could be mapped to the spare cells using couple of ways. One way could be, for each clock buffer, check the physical location of the nearest available spare buffer cell and map them. Other way could be to use the available tool which is capable of mapping a design to the spare cells.

In this project, different options were tried and came up with final optimal flow. There were two iterations in this flow. In the first iteration (Figure 18), the clock tree generating tool [22] would be allowed to generate the clock tree by adding buffers from the library. This would change the base layer; however, the tool would be allowed to violate this constraint in this iteration. The type of buffer cells that could be used to implement the clock tree would be given by the designer from the library. Given the flexibility to use buffer cells, the CTS tool would be able to implement a better clock tree with lesser skew as compared to the previous experiment. After generating the clock tree, the modified netlist would be saved with all the newly added clock buffers. The only tool which had the capability to exhaustively use the spare cells was the Conformal ECO. Because of this enabling feature, the Conformal ECO was chosen as a candidate to map the newly added clock buffers to the spare cells. In the second iteration (Figure 19) of this developed flow, the netlist (with the clock buffers) generated from the first iteration

would be given to the Conformal ECO as the revised netlist. The golden netlist would be the same feedstock-based netlist with no connectivity (as discussed in section 2.3.5). The goal of the Conformal ECO would be to implement the revised netlist using the spare cells present in the golden netlist. The same Conformal ECO flow, as discussed in section 2.3.5 would be run to achieve this goal.

After the Conformal ECO would complete implementation of the design, the post-mask placement EDI flow (as discussed in section 3.1) would be run. The EDI flow would physically map all the cells used to implement the design, to the spare cells. After the cells were all placed, the CTS would be run again to generate the clock tree. However, since the clock buffers were already present in the design, routing would be sufficient for generating the final clock tree. This process would thus enable the generation of a clock tree with buffers.

3.2.3 Retaining Clock Buffers in Conformal ECO

In the newly generated Conformal ECO based CTS flow, it was assumed that the clock buffers inserted by the CTS tool during the first iteration would not be removed by the Conformal ECO. However, there was a possibility that the clock buffers would get removed during the Conformal ECO stage because the Conformal ECO does logical equivalence while comparing the golden and revised netlist.

An experiment was run to check if this assumption was valid. In this experiment, a sample design was taken and the clock tree was generated using the standard CTS flow. Then, the modified netlist was given back to the Conformal ECO to check if it retained the clock buffers. It was found that the Conformal ECO removed all the clock buffers from the patch file. None of the clock buffers were thus, present in the final netlist.

In order to keep the clock buffers in the netlist, a workaround was conceived by defining the clock buffers as black boxes. This would prevent the Conformal ECO from removing the clock buffers, thus retaining them in the final netlist. The feasibility of this approach was tested in an experiment where all the clock buffers were defined as black boxes in the Conformal ECO. Upon running the Conformal ECO, it was found that in spite of defining the clock buffers as black boxes, the clock buffers were being removed from the final netlist.

A more careful investigation revealed that that the Conformal ECO was indeed retaining the clock buffers, defined as black boxes, in the netlist. However, the RC engine of the Conformal ECO retained the visibility of the functionality of the clock buffers and hence, removed them. Retention of the clock buffers in the RC was achieved by removing the functionality of the clock buffers from the '.lib' itself. In this way, the RC did not have any knowledge of the functionality of the clock buffers and hence, retained the clock buffers in the final netlist.

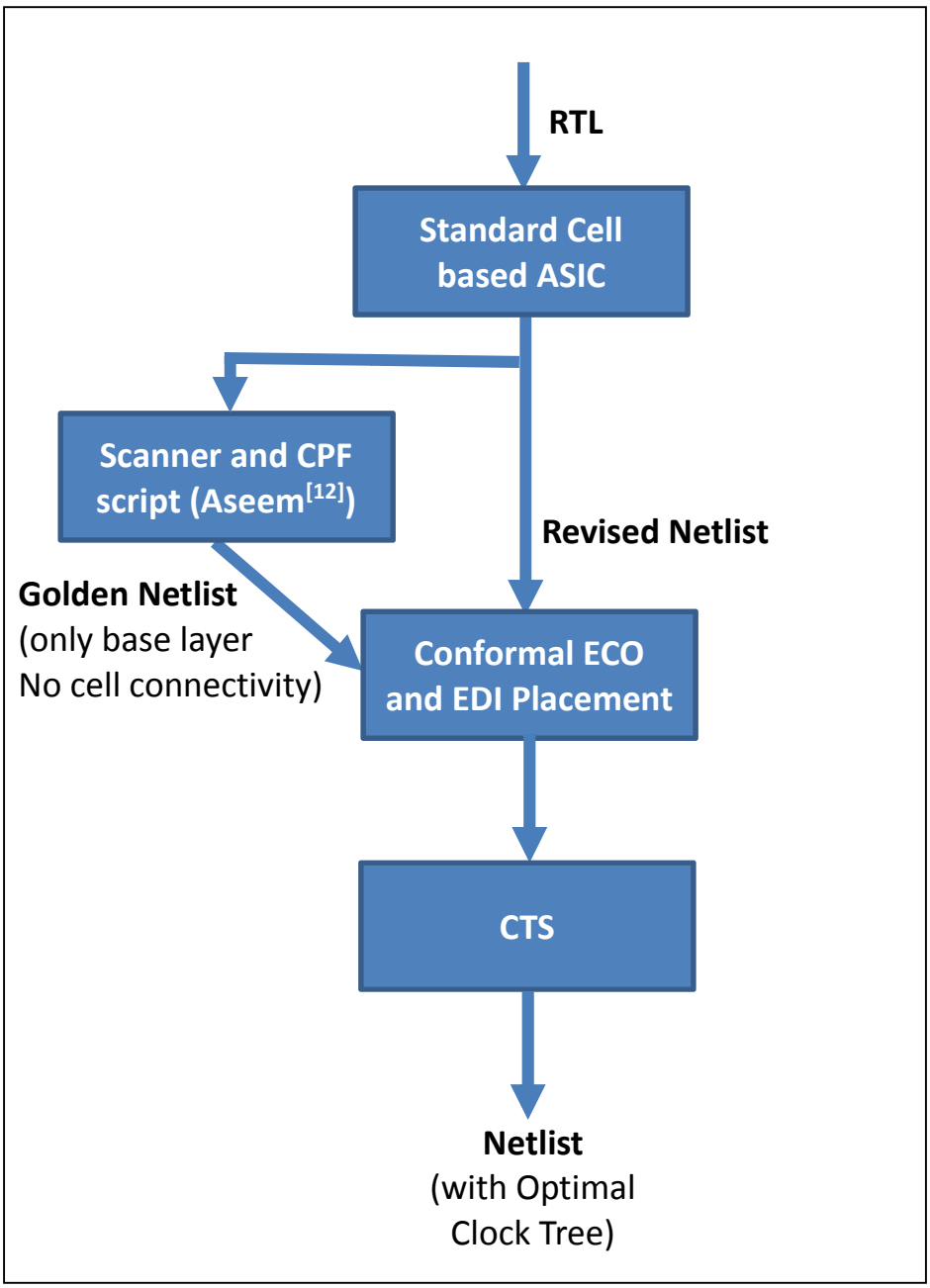


Figure 18: Developed Post Mask CTS Flow - Iteration 1

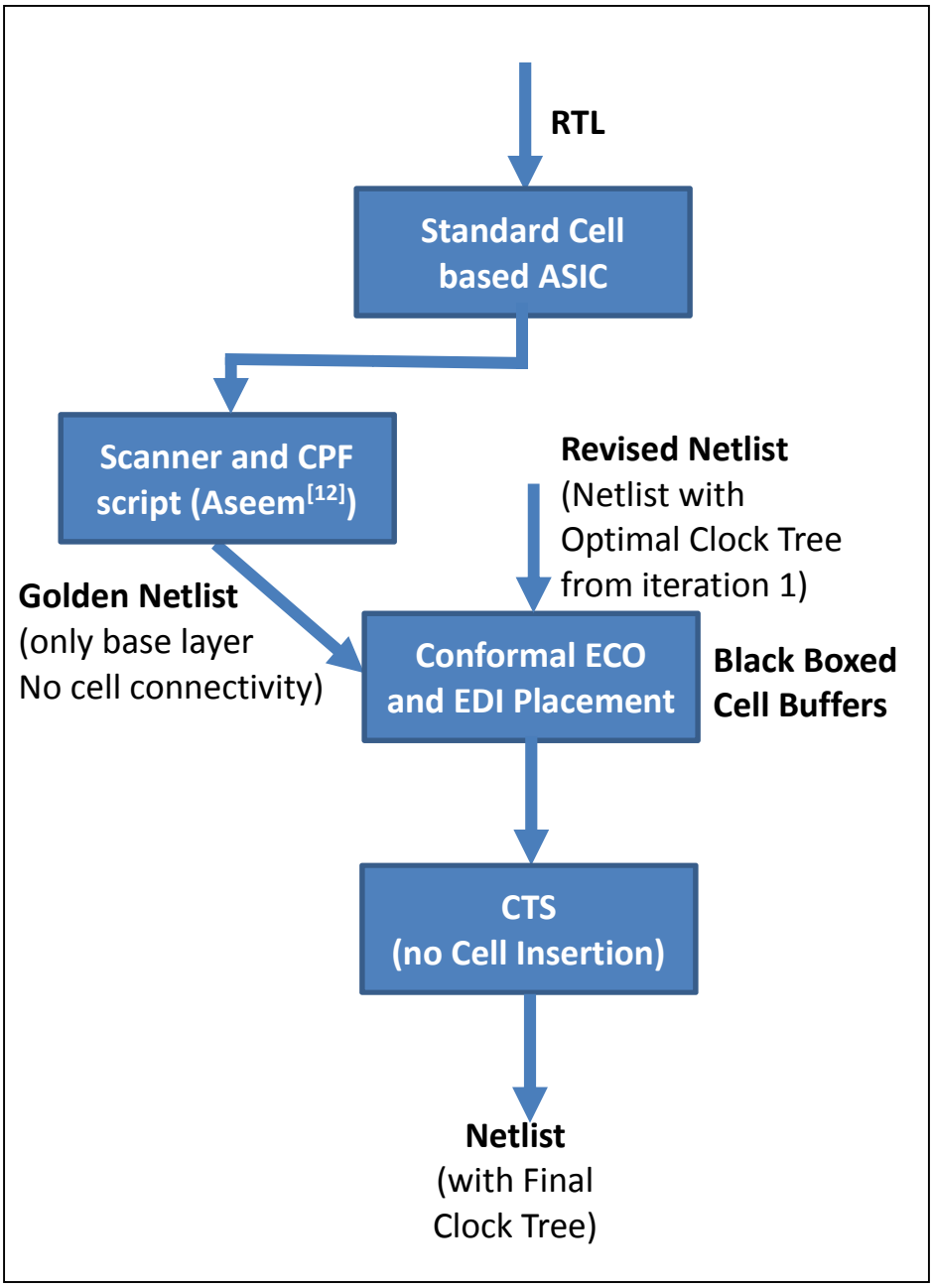


Figure 19: Developed Post Mask CTS Flow - Iteration 2

3.2.4 Optimization Techniques

Additional optimization techniques were attempted to further improve the developed flow. While developing the clock tree in the first iteration, there was an option to make the flop cells movable. While this would improve the quality of the generated clock tree after the first iteration, it could also deteriorate the final clock tree at the end of the second iteration since the base layer is fixed. Hence, experiments were run with and without exercising the option of moving the flop cells. The clock trees generated after the first iteration are shown in Figure 20 and 21.

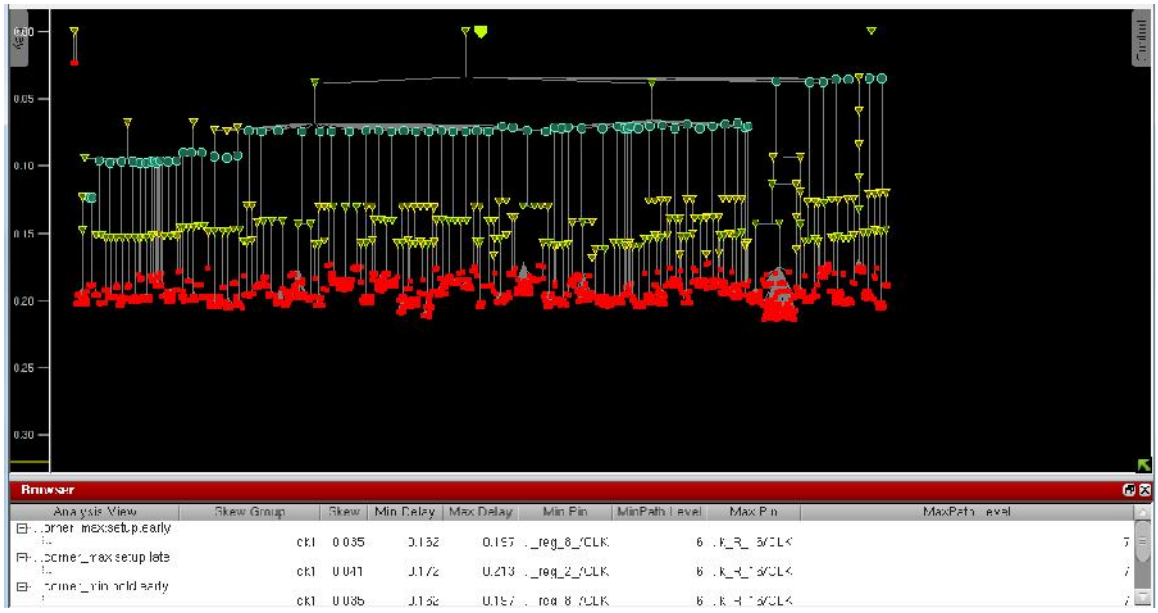


Figure 20: Built Clock Tree (iteration 1) with Non-Movable Flops

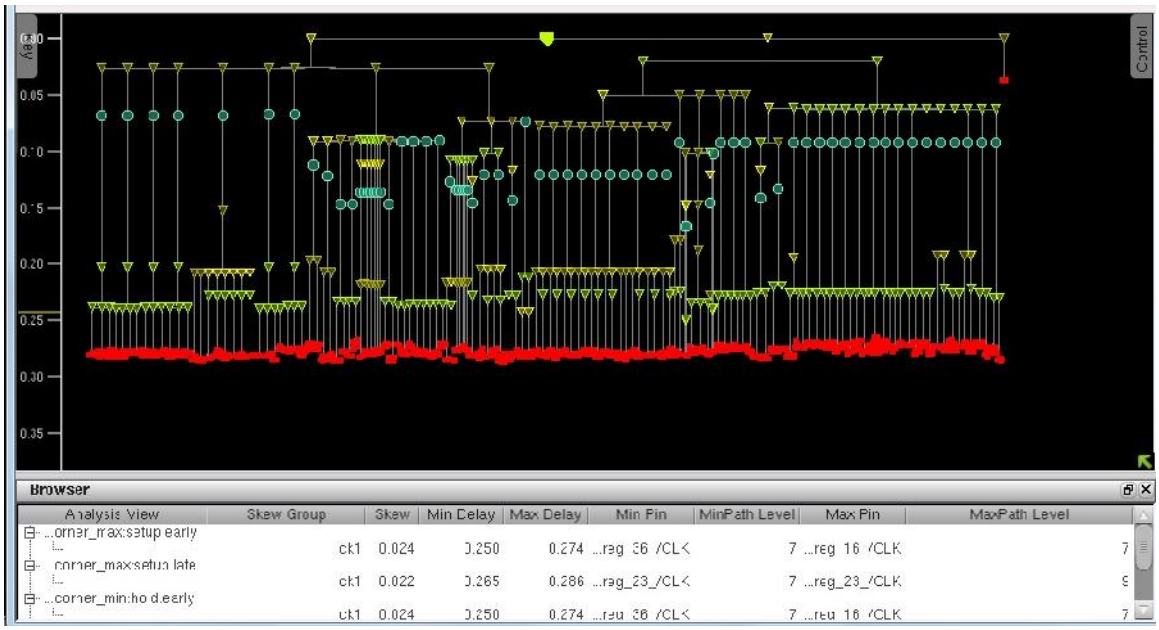


Figure 21: Built Clock Tree (iteration 1) with Movable Flops

A second optimization technique was to use the output netlist of the Conformal ECO (section 2.3.5) as the golden netlist in the second iteration of the developed flow as this output netlist had no modification in the base layer. Experiments were run where instead of the feedstock-based netlist (with no connectivity), the Conformal ECO output netlist was given as the golden netlist. The final clock trees generated from this technique are given below.

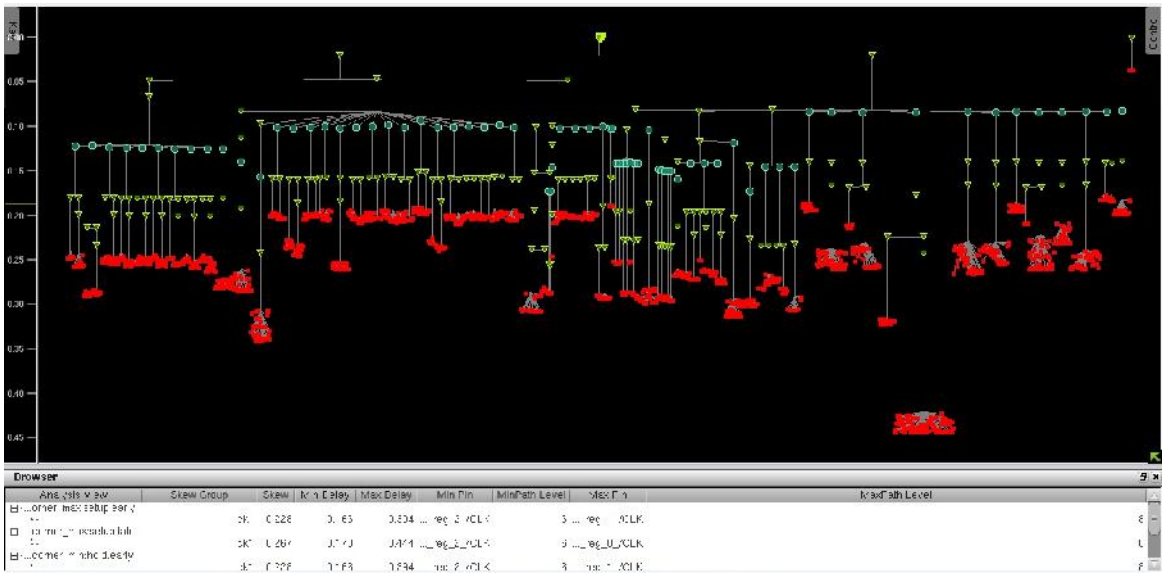


Figure 22: With Unconnected Feedstock Netlist, as Golden Netlist

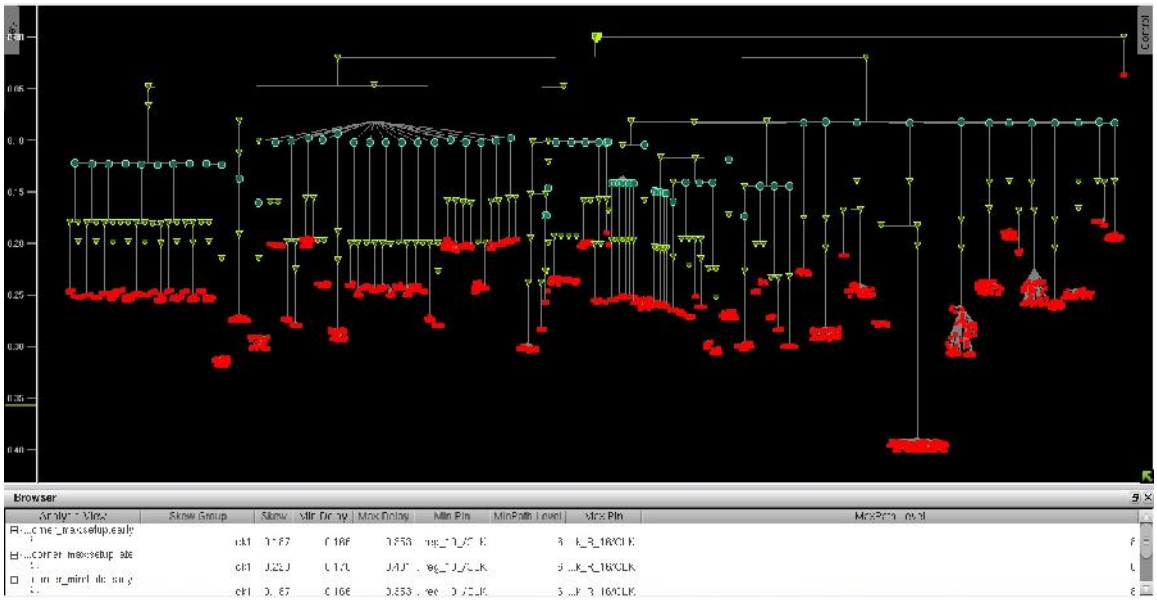


Figure 23: With Conformal ECO Netlist, as Golden Netlist

3.2.5 Results

In this section, results from implementation of a clock tree for the Amber SOC single core design with the developed 'CTS with Conformal ECO' flow, are presented. The goal of this experiment was to analyze the quality of the clock tree implemented by this flow. Table 1 compares the clock tree for different optimization techniques.

		Max Skew (ns)	Max Delay (ns)
Clock tree using only Routing (Figure 17)		0.382	0.385
1st iteration (Standard CTS)	With Non - Movable Flops (Figure 20)	0.041	0.213
	With Movable Flops (Figure 21)	0.022	0.286
2nd iteration (Final generated clock tree)	With Unconnected Feedstock netlist as Golden netlist (Figure 22)	0.267	0.444
	With Conformal ECO netlist as golden netlist (Figure 23)	0.223	0.401

Table 1: Clock Tree Quality Metric for Different Optimizations

It can be seen that the clock tree generated using only routing performed poorly with a maximum skew of 0.382 ns. On the other hand, the clock tree obtained after the first iteration had a skew of 0.022 ns, which was set as a goal for the second iteration. After running the second iteration, the maximum skew of the clock tree was 0.223 ns. Clearly, it was not as good as the standard clock tree. However, given the constraint of a

fixed base layer, the developed flow performed better than the use of only routing to implement the clock tree.

The clock tree could be further improved by planning for clock tree synthesis while generating feedstocks itself. There are different options that could be tried. One such way could be to have 'mini' clock tree in each feedstock itself. Once the feedstocks are stitched together it would generate the main clock tree. Also, more advanced features could be added such that it would generate the main clock tree with zero skew. For example, all feedstocks should add same load on the main clock tree, there could be multiple 'inout' clock pins in each feedstock; say on all four sides of the feedstock so that the main clock can enter the feedstock from any side.

Other way, of CTS aware feedstock generation, could be to have 'intentional' sites for clock buffers in the feedstocks. These sites could have couple of clock buffers of different drive strength and V_T. Now CTS tool could be constrained such that it inserts clock buffers at these sites only. This could be achieved in different ways, say by inserting blockages at all other places except these sites. Once these advanced features are incorporated in the feedstocks, the would have very promising results.

3.3 TIMING CLOSURE – POST MASK

In a typical ASIC flow, the timing of the design is checked after the design is placed and routed. The timing closure stage of the ASIC design is used to address the challenging task of meeting the timing requirement of the design. First, the RC values are extracted for the current design and the timing engine is run to get timing numbers for each path. The design is incrementally optimized for paths that do not meet the timing numbers. This is done by adding buffer cells, changing drive strength of the existing cells

and V_T swapping. Since these incremental changes involve adding/modifying cells, the base layer of the design gets modified.

However, the M2A2 design flow constrained the base layer to not be modified. Hence, a variety of experiments were run to check whether the existing flow was capable to perform timing optimization in post-mask mode, and to capture the requirements of a new flow. The tool used for timing analysis and incremental optimization was Tempus. Based on the design modifications suggested by Tempus, the physical design was done in Innovus.

3.3.1 Timing Closure Using Tempus and Innovus

In this step, the basic timing closure flow was first setup. Two flows were available to fix timing, one in Innovus and second using Tempus and Innovus combined. The Innovus based flow used the 'optDesign' command to fix timing. In this flow, there was an option to make the timing optimization as post mask, by disabling all the knobs which would allow the tool to modify the base layer. However, there would not be much improvement in timing through this way.

In the Tempus and Innovus combined flow [24], first the RC extraction was done in Innovus along with the generation of a 'spef' file. This file provided the resistor and capacitor values of each net. The constraint file, library and lefs were also input. Tempus was then run to analyze the setup and hold failures, and the critical paths. A dump of the database with the timing analysis results was created, which was used to analyze the critical paths and incremental optimizations. The configuration file for timing optimization was then defined, which involved specifying the buffer and load cells to be used during optimization and whether cell sizing or V_T cell swapping was allowed,

among other parameters. Once this setup was ready, Tempus was run to fix the timing and Design Rule Violations (DRVs).

After the optimization was done, it generated the ECO script listing all the incremental changes needed in the design to fix timing and DRVs. This script was loaded in Innovus, enabling the addition or modification of the cells in the design. Then, ECO routing was done to connect all the newly added cells. Finally, the RC netlist was extracted and timing analysis done again to check whether timing and DRVs were fixed. The flow diagram is shown in the Figure 24.

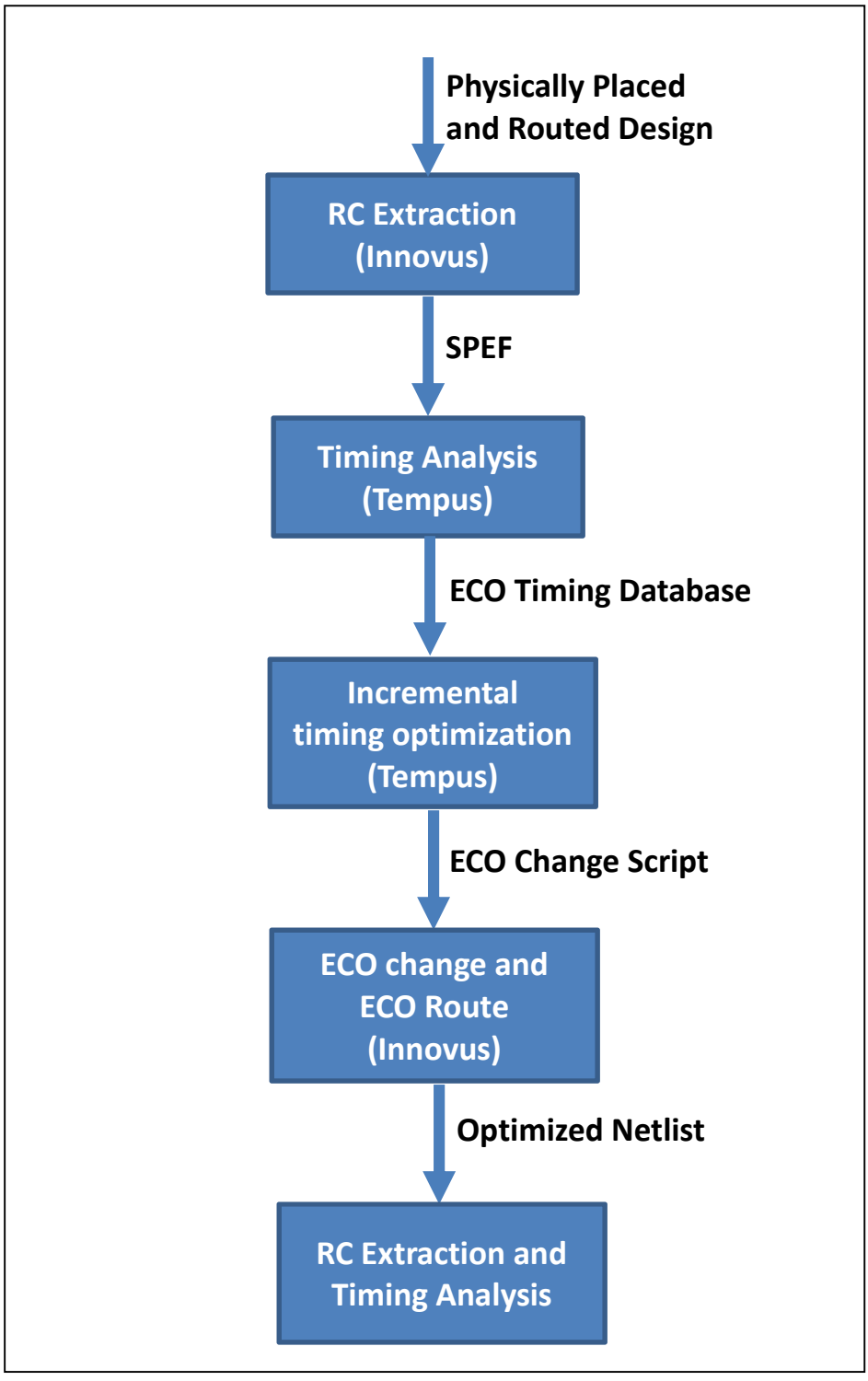


Figure 24: Timing Closure Flow Using Tempus and Innovus [24]

3.3.2 Limitations of the Standard Flow

Both the timing closure flow (using only Innovus, Innovus and Tempus combined) discussed above had many features like buffer insertion, load cell insertion, gate sizing, V_T cell swapping etc., to close timing. However, there was no 'exhaustive' cell usage algorithm in these tools, which limited the tools from conducting optimization by using only the spare cells. Hence, a new flow had to be developed, which would use only the spare cells and optimize the design.

3.3.3 Developing the New Flow

The concept for this new flow was borrowed from the 'CTS with Conformal ECO' flow (Section 3.2.2). The main idea of this flow was that two iterations would be run since the tool did not have the feature of optimizing the design using only spare cells. In the first iteration, the tool would be allowed to add new cells to the design. This would allow the use of all available advanced timing closure techniques of the tool. After the tool had optimized the design, a dump of the final netlist would be created. This netlist would be given to the Conformal ECO which has the 'exhaustive' cell usage algorithm. Hence, in this way, it would be possible to map the new cells added during the incremental changes in the first iteration to the spare cells already present in the design. The Post Mask timing closure flow is shown in Figure 25.

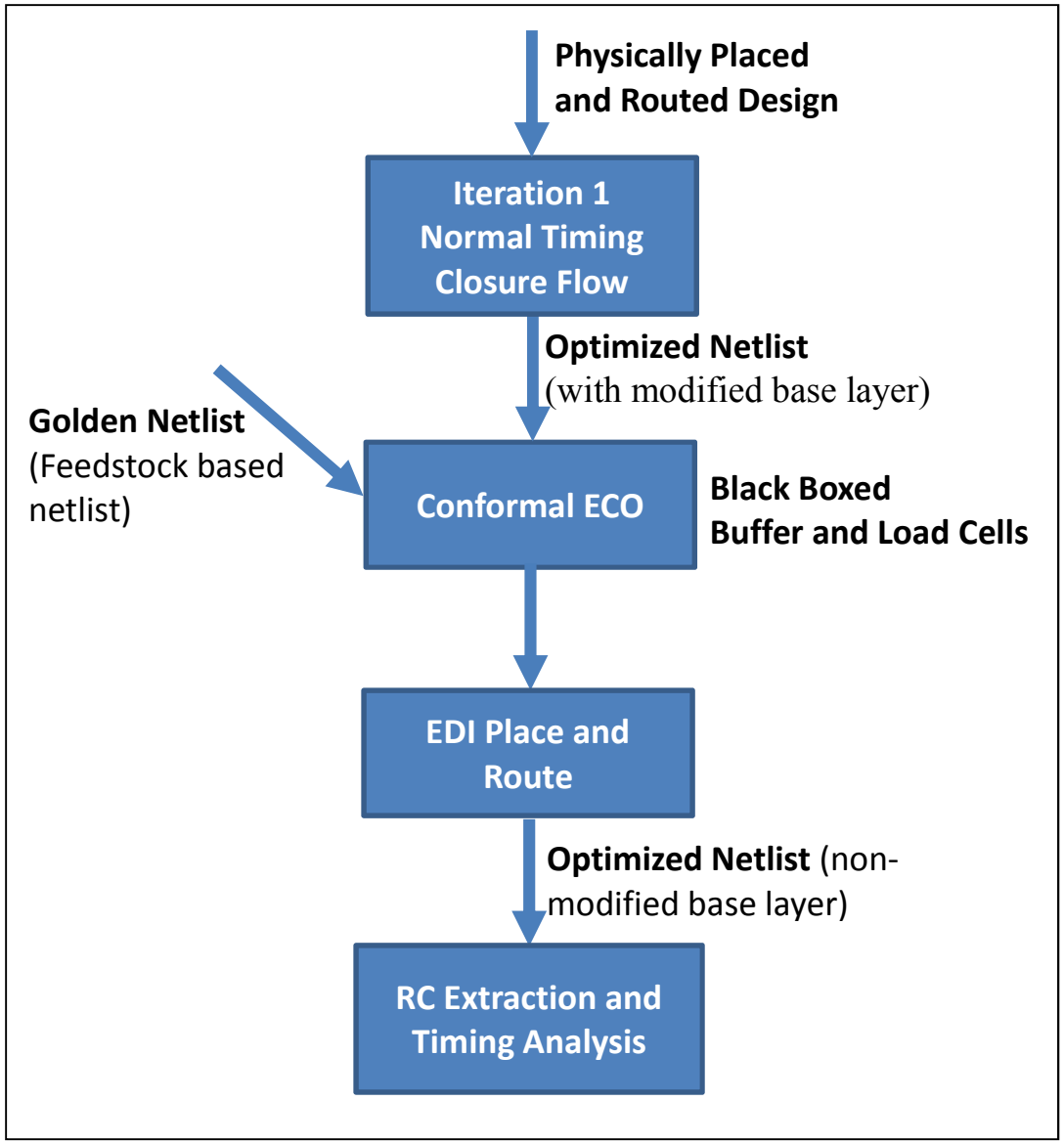


Figure 25: Develop Post Mask Timing Closure Flow

3.3.4 Optimization Technique – Empty vs Conformal ECO netlist

In order to improve the flow further, some additional optimization techniques were tried. One technique involved the use of the Conformal ECO output netlist (section 2.3.5) as the golden netlist in the second iteration of the developed flow. Since this output netlist also had no modification in the base layer, it was a good candidate for the golden netlist. It was found that with this optimization, the Conformal ECO had to only compare

the incremental changes, instead of mapping the whole design to the spare cells. This allowed it to implement a better design.

3.3.5 Black Boxing Buffer Cells of a Particular Drive Strength

A second optimization technique involved defining particular types of buffer cells (for example, high drive strength buffer cells) as black boxes. By black boxing a type of buffer cell, a hard constraint that the buffer cell had to be mapped to a spare cell, was put. For example, if BUFFX32 was defined as a black box and used to optimize the design, the Conformal ECO would search for a spare BUFFX32 in the design. If the Conformal ECO was not able to search in the nearby location, then it was constrained to pick from a location further away. Because the cell was a black box, it could not be ignored. The advantage, however, was that the Conformal ECO did not remove this buffer from the final netlist during its internal optimization.

As can be seen, this optimization involved a trade-off which might improve or even deteriorate the final design. Hence, different iterations should be used to check for the best results. For example, if there are an adequate number of spare buffers in the design, then all should be black boxed. However, if there is an area restriction, then the number of spare buffers of a particular type should be increased and defined as a black box.

Drive Strength of Buffers which were Black Boxed	WNS (Normalized)	TNS (Normalized)
Low Drive Strength (2x, 4x, 8x)	-1.33	-1.43
High Drive Strength (8x, 16x, 32x)	-1	-1
All Drive Strength (2x, 4x, 8x, 16x, 32x)	-1.21	-1.06

Table 2: Timing comparison for Different Black Boxed Buffers

This process was validated with an experiment run on the Amber SOC design. Different types of buffers were black boxed and consequent timing numbers measured. The results are summarized in Table 2. It can be clearly observed that black boxing only the low drive strength buffer cells gave the worst timing numbers. This could be explained by the fact that the extra time needed to reach the low drive strength buffer cell was more than the time saved by that buffer. Hence, low drive strength buffer cells should be black boxed only when they are present in sufficient numbers. The best performance was achieved when only the high drive strength buffer cells were black boxed. As high drive strength buffers were crucial for high fanout nets, it should be ensured that Conformal ECO does not remove them during the spare cell mapping stage. Hence, black boxing the high drive strength buffer cells resulted in better performance.

3.3.6 Optimizing Critical Paths Iteratively

Since the performance of this Post Mask timing closure flow described above depends on the number of spare buffers in the design, it was important to use this scarce

resource optimally. To this end, an optimization technique was tried towards first optimizing the most critical paths. In this technique, the Post Mask timing flow would be run iteratively with each iteration optimizing a particular number among the worst timing paths.

The Tempus flow would be run with the objective of optimizing those paths and generating the ECO script. This script would be given back to the Conformal ECO for mapping the newly added buffer cells to the available spare cells. The timing analysis of the design would be done again to get the new critical timing paths. This process would be iterated until the timing of all the paths was fixed or when there were no spare buffer cells remaining in the design. This would enable the more critical paths to get priority and have access to a larger number of spare buffer cells compared to the less critical paths.

3.3.7 Results

The developed Post Mask timing closure flow was run on the Amber SOC dual core high performance design to validate any improvement in timing violations. The critical path before and after running this flow was compared. The initial feedstock-based design was not optimized and had some very high fanouts of 100 or more. The following figure shows a net which had a fanout of 124 with the 124 cells spread across the entire design. This net was being driven by an inverter.

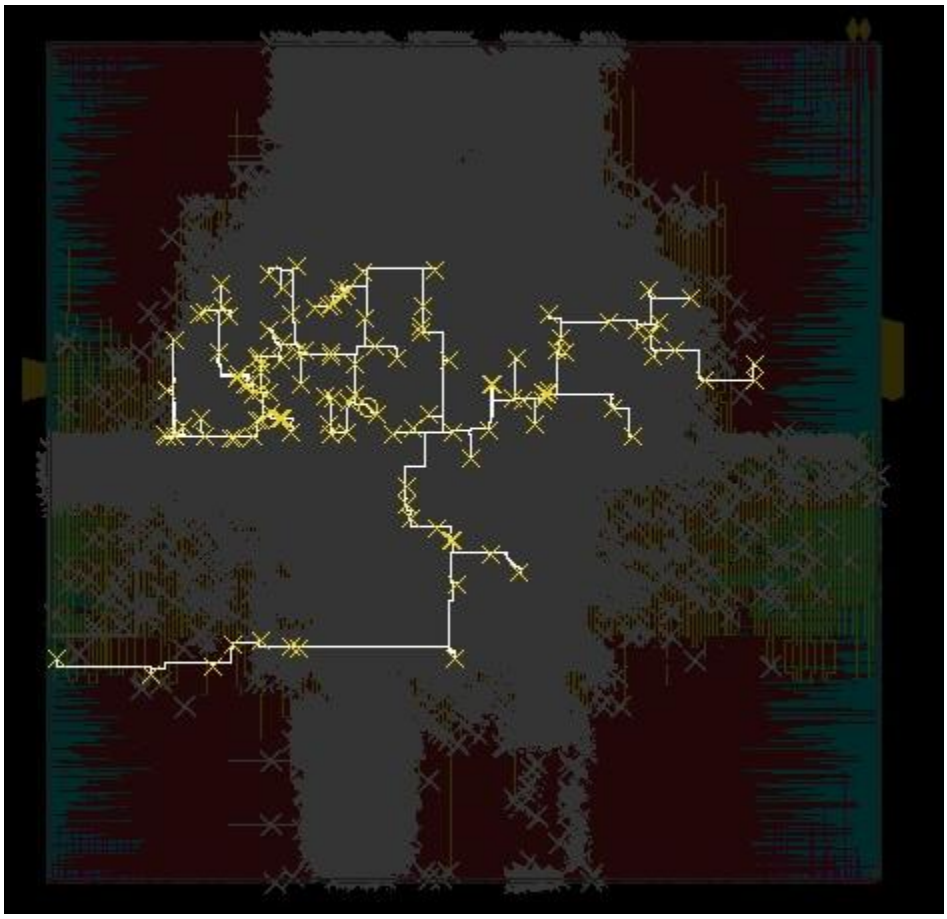


Figure 26: Sample Net in the Initial Feedstock Based Design with fanout of 124

Since the initial design was not optimized, it had a critical slack of -65.94 ns. After running the developed Post Mask timing closure flow, the slack of this particular path improved to -19.98 ns, but another path became the new critical path with a slack of -40.65 ns. The snapshots of the critical paths are given below.

```
Startpoint: ECO2reg_u_core_output_mux_count_reg_3_
            (rising edge-triggered flip-flop clocked by sys_clk)
Endpoint: ECO2reg_u_amber0_u_decode_pre_fetch_instruction_reg_17_
          (rising edge-triggered flip-flop clocked by sys_clk)
Path Group: sys_clk
Path Type: max

Point                                                 Cap      Incr       Path
-----------------------------------------------------------------------------
clock sys_clk (rise edge)                                      0.00       0.00
clock network delay (ideal)                                    0.00       0.00
ECO2reg_u_core_output_mux_count_reg_3_/CLK (DFFX2_LVT)
                                                               0.00       0.00 r
ECO2reg_u_core_output_mux_count_reg_3_/Q (DFFX2_LVT)
                                                      12.98    0.09       0.09 f
ECO2inst_32926/Y (NAND3X0_RVT)                         0.55    1.07       1.15 r
ECO2inst_32925/Y (NAND3X0_LVT)                         0.58    0.18       1.33 f
ECO2inst_31440/Y (NBUFFX2_LVT)                         9.01    0.20       1.54 f
ECO2inst_31442/Y (INVX8_LVT)                          20.38    3.65       5.18 r
ECO2inst_28097/Y (INVX2_LVT)                          15.89    4.20       9.39 f
ECO2inst_10691/Y (INVX4_LVT)                          29.21    7.34      16.73 r
ECO2inst_8450/Y (MUX21X1_LVT)                          0.47    4.56      21.29 f
ECO2inst_7009/Y (AND3X1_LVT)                           1.23    0.17      21.46 f
ECO2inst_6457/Y (NAND4X0_LVT)                          0.60    0.23      21.69 r
ECO2inst_6014/Y (OA21X1_LVT)                           1.32    0.20      21.90 r
ECO2inst_5929/Y (OR2X1_LVT)                            2.72    0.28      22.18 r
ECO2inst_27158/Y (AO21X1_LVT)                          1.66    0.25      22.43 r
ECO2inst_39926/Y (AND2X1_RVT)                          0.64    0.26      22.69 r
ECO2inst_39925/Y (NAND3X0_LVT)                         1.24    0.21      22.89 f
ECO2inst_39927/Y (AND2X1_LVT)                          1.95    0.25      23.14 f
ECO2inst_5427/Y (AND2X1_LVT)                           1.66    0.28      23.42 f
ECO2inst_5234/Y (AOI21X1_RVT)                          1.36    0.27      23.69 r
ECO2inst_4954/Y (AO22X2_LVT)                           1.91    0.24      23.92 r
ECO2inst_29721/Y (INVX0_RVT)                           0.57    0.26      24.18 f
ECO2inst_29720/Y (AO21X1_LVT)                          1.18    0.20      24.38 f
ECO2inst_29719/Y (NAND2X0_LVT)                         0.63    0.22      24.59 r
ECO2inst_4897/Y (AND2X2_LVT)                           1.21    0.21      24.80 r
ECO2inst_32492/Y (AND2X1_RVT)                          1.44    0.25      25.05 r
ECO2inst_39320/Y (OR2X2_LVT)                           2.83    0.25      25.31 r
ECO2inst_39319/Y (INVX4_LVT)                           4.79    0.84      26.15 f
ECO2inst_39318/Y (AND2X4_LVT)                         14.99    0.50      26.64 f
ECO2inst_26805/Y (INVX4_LVT)                          25.82    6.23      32.87 r
ECO2inst_26810/Y (INVX16_LVT)                         82.79   26.71      59.58 f
ECO2inst_24256/Y (NBUFFX4_LVT)                        33.93    3.80      63.38 f
ECO2inst_24255/Y (INVX1_HVT)                           0.72    3.37      66.75 r
ECO2inst_22987/Y (NOR2X1_LVT)                          0.56    0.24      66.99 f
ECO2reg_u_amber0_u_decode_pre_fetch_instruction_reg_17_/D (DFFX1_RVT)
                                                               0.16      67.15 f
data arrival time                                                        67.15

clock sys_clk (rise edge)                                      1.25       1.25
clock network delay (ideal)                                    0.00       1.25
clock uncertainty                                             -0.02       1.23
ECO2reg_u_amber0_u_decode_pre_fetch_instruction_reg_17_/CLK (DFFX1_RVT)
                                                               0.00       1.23 r
library setup time                                            -0.02       1.21
data required time                                                       1.21
-----------------------------------------------------------------------------
data required time                                                       1.21
data arrival time                                                      -67.15
-----------------------------------------------------------------------------
slack (VIOLATED)                                                       -65.94
```

Figure 27: Critical Path in the Initial Design

```
Startpoint: ECO2reg_ECO2reg_u_core_output_mux_count_reg_3_
            (rising edge-triggered flip-flop clocked by sys_clk)
Endpoint: ECO2reg_ECO2reg_u_amber0_u_decode_pre_fetch_instruction_reg_17_
          (rising edge-triggered flip-flop clocked by sys_clk)
Path Group: sys_clk
Path Type: max

Point                                                 Cap      Incr       Path
-----------------------------------------------------------------------------
clock sys_clk (rise edge)                                      0.00       0.00
clock network delay (ideal)                                    0.00       0.00
ECO2reg_ECO2reg_u_core_output_mux_count_reg_3_/CLK (DFFX2_LVT)
                                                               0.00       0.00 r
ECO2reg_ECO2reg_u_core_output_mux_count_reg_3_/Q (DFFX2_LVT) <-
                                                      5.40     0.07       0.07 f
ECO2inst_plcwnsbuf_st5077481/Y (NBUFFX4_LVT)
                                                     19.05     0.46       0.54 f
ECO2inst_plcwnsbuf_st5077466/Y (INVX4_LVT)
                                                     17.18     6.70       7.23 r
ECO2inst_g5056965/Y (MUX21X1_LVT)                     0.54     2.68       9.91 r
ECO2inst_g5056964/Y (AO22X1_LVT)                      1.79     0.18      10.09 r
ECO2inst_g989815/Y (AND2X1_LVT)                       1.13     0.25      10.34 r
ECO2inst_g988600/Y (NAND3X0_LVT)                      1.50     0.20      10.54 f
ECO2inst_g988207/Y (OR2X1_LVT)                        4.43     0.26      10.79 f
ECO2inst_g988202/Y (INVX1_LVT)                        2.88     0.47      11.26 r
ECO2inst_g987294/Y (NAND3X0_LVT)                      0.62     0.28      11.55 f
ECO2inst_g986892/Y (AND2X1_LVT)                       0.47     0.19      11.73 f
ECO2inst_g986577/Y (AND3X1_LVT)                       1.30     0.16      11.89 f
ECO2inst_g986576/Y (INVX0_LVT)                        1.86     0.21      12.10 r
ECO2inst_g4872040/Y (AND2X1_LVT)                      3.89     0.25      12.35 r
ECO2inst_g5025441/Y (AND2X1_LVT)                      1.80     0.38      12.73 r
ECO2inst_g6989181/Y (AND2X1_LVT)                      2.62     0.25      12.98 r
ECO2inst_g5025445/Y (AO22X1_LVT)                      1.71     0.26      13.24 r
ECO2inst_g5025424/Y (OA22X1_RVT)                      2.35     0.24      13.49 r
ECO2inst_g5025427/Y (NAND2X0_LVT)                     0.58     0.21      13.69 f
ECO2inst_g5025426/Y (OA21X1_LVT)                      1.31     0.18      13.88 f
ECO2inst_g984257/Y (OR2X1_RVT)                        0.62     0.25      14.13 f
ECO2inst_g5950889/Y (AND2X1_LVT)                      1.25     0.19      14.31 f
ECO2inst_g5950888/Y (AND2X1_LVT)                      2.10     0.22      14.54 f
ECO2inst_plcwnsbuf_st5058438/Y (INVX1_RVT)
                                                      1.11     0.30      14.83 r
ECO2inst_g9160695/Y (NAND2X4_LVT)                     3.63     0.22      15.06 f
ECO2inst_g9160699/Y (INVX4_LVT)                       5.90     0.91      15.97 r
ECO2inst_g9161771/Y (NAND2X4_LVT)                     7.68     0.34      16.31 f
ECO2inst_plcwnsbuf_st9161773/Y (NBUFFX8_LVT)
                                                      6.07     1.13      17.44 f
ECO2inst_g5241851/Y (INVX4_LVT)                      33.37     1.46      18.90 r
ECO2inst_g983840/Y (AND2X1_HVT)                       0.56     2.10      21.00 r

                                                               0.15      21.14 r
data arrival time                                                        21.14

clock sys_clk (rise edge)                                      1.25       1.25
clock network delay (ideal)                                    0.00       1.25
clock uncertainty                                             -0.02       1.23

                                                               0.00       1.23 r
library setup time                                            -0.06       1.17
data required time                                                       1.17
-----------------------------------------------------------------------------
data required time                                                       1.17
data arrival time                                                      -21.14
-----------------------------------------------------------------------------
slack (VIOLATED)                                                       -19.98
```

Figure 28: Same Path After Timing Optimization

```
Startpoint: ECO2reg_ECO2reg_u_amber1_u_decode_o_barrel_shift_function_reg_1_
            (rising edge-triggered flip-flop clocked by sys_clk)
Endpoint: ECO2reg_ECO2reg_u_amber1_u_execute_u_register_bank_r11_reg_31_
          (rising edge-triggered flip-flop clocked by sys_clk)
Path Group: sys_clk
Path Type: max

Point                                                    Cap      Incr       Path
-----------------------------------------------------------------------------------
clock sys_clk (rise edge)                                         0.00       0.00
clock network delay (ideal)                                       0.00       0.00

                                                                  0.00       0.00 r

                                                         2.99     0.09       0.09 f
ECO2inst_g321/Y (NAND2X4_LVT)                           31.09     0.30       0.40 r
ECO2inst_g4957897/Y (INVX16_LVT)                        31.03    28.94      29.34 f
ECO2inst_g4940334/Y (AND2X1_LVT)                         3.49     1.92      31.26 f
ECO2inst_g979839/Y (INVX1_RVT)                           2.91     0.42      31.68 r
ECO2inst_g972244/Y (OR2X1_RVT)                           0.56     0.32      32.00 r
ECO2inst_g972189/Y (AND2X1_RVT)                          0.62     0.18      32.18 r
ECO2inst_g971310/Y (AND2X1_RVT)                          0.62     0.19      32.37 r
ECO2inst_g970495/Y (AND2X1_LVT)                          0.56     0.18      32.56 r
ECO2inst_g970399/Y (AND2X1_LVT)                          0.57     0.17      32.72 r
ECO2inst_g317/Y (AND3X1_LVT)                             0.57     0.17      32.90 r
ECO2inst_g314/Y (AND3X1_LVT)                             0.69     0.17      33.07 r
ECO2inst_g311/Y (NAND4X0_LVT)                            1.10     0.21      33.28 f
ECO2inst_g969840/Y (MUX21X1_LVT)                         1.68     0.21      33.49 f
ECO2inst_g9161465/Y (NAND3X0_LVT)                        1.19     0.25      33.74 r
ECO2inst_g402/Y (OR2X2_LVT)                              3.81     0.25      33.99 r
ECO2inst_g401/Y (INVX2_LVT)                              2.26     0.68      34.67 f
ECO2inst_g5075654/Y (NAND4X0_LVT)                        2.06     0.31      34.98 r
ECO2inst_g5076194/Y (OR3X1_LVT)                          3.27     0.25      35.23 r
ECO2inst_g5077152/Y (XOR2X1_LVT)                         1.88     0.49      35.72 r
ECO2inst_g5077156/Y (INVX1_LVT)                          1.08     0.30      36.02 f
ECO2inst_g5077157/Y (AO22X1_LVT)                         2.45     0.20      36.22 f
ECO2inst_g968974/Y (INVX1_LVT)                           1.22     0.33      36.56 r
ECO2inst_g182/Y (NAND3X0_LVT)                            1.24     0.21      36.77 f
ECO2inst_g214/Y (NAND4X0_LVT)                            1.13     0.25      37.02 r
ECO2inst_g5059143/Y (NAND3X0_LVT)                        1.58     0.20      37.21 f
ECO2inst_g5059147/Y (NAND3X0_LVT)                        0.64     0.22      37.43 r
ECO2inst_g968785/Y (NAND3X0_LVT)                         1.22     0.19      37.62 f
ECO2inst_g623380/Y (NAND3X0_LVT)                         0.53     0.22      37.84 r
ECO2inst_g618502/Y (AND3X1_LVT)                          0.68     0.17      38.01 r
ECO2inst_g614004/Y (OR3X2_LVT)                           2.83     0.21      38.23 r
ECO2inst_eco_buf5319587/Y (INVX4_LVT)                    5.51     0.76      38.99 f
ECO2inst_g5937924/Y (INVX8_LVT)                         13.17     1.47      40.46 r
ECO2inst_g5721344/Y (AO22X1_LVT)                         0.56     1.24      41.70 r
ECO2reg_ECO2reg_u_amber1_u_execute_u_register_bank_r11_reg_31_/D (DFFX1_RVT)
                                                                  0.15      41.85 r

clock sys_clk (rise edge)                                         1.25       1.25
clock network delay (ideal)                                       0.00       1.25
clock uncertainty                                                -0.02       1.23

                                                                  0.00       1.23 r
library setup time                                               -0.03       1.20
data required time                                                           1.20
-----------------------------------------------------------------------------------
data required time                                                           1.20
data arrival time                                                          -41.85
-----------------------------------------------------------------------------------
slack (VIOLATED)                                                           -40.65
```

Figure 29: Critical Path in the Optimized Design

Hence, it can be seen that this technique successfully reduced the critical slack from -65.94ns to -40.65ns. However, the final slack was still undesirably poor. The number of available spare buffers plays a crucial role. Since these spare buffers would be pre-placed in the design, the availability of a space buffer in the near vicinity could be ensured by having an adequate number of spare buffers distributed throughout the design.

Table 3 summarizes the timing slack, worst negative slack(WNS) and total negative slack(TNS), after different optimizations for the High Performance CDL 50 feedstock based design, developed by one of the project collaborator [12].

High Performance CDL 50	**Pre-CTS – Placement Aware (WNS/TNS)**	**Post-Route (WNS/TNS)**
Original Design	-35.74 / -103,000 ns	-55.42 / -146,000 ns
Fixed Base Layer Optimization in Innovus	-23.79 / -86,034 ns	-38.85 / -126,000 ns
After optimizations in Tempus	-25.77 / -45,546.1 ns	-47.89 / -82,953 ns
Fixed Base Layer Optimization in Innovus (after Optimizations in Tempus)	-21.24 / -40,178.2 ns	-54.46 / -82,457 ns

Table 3: Timing Slack for High Performance CDL 50

It could be seen that the Post-Route critical slack improved from -55.42 ns to -38.85 ns in Innovus. Also in Tempus, although the critical slack improved to -47.89 ns only but the TNS improved by roughly 50%. In this experiment, all the timing paths were optimized in one iteration. Hence all the timing paths got equal preference, leading to great improvement in the total negative slack. However, as discussed in section 3.3.5, the most critical paths could be given preferences by running multiple iterations.

Chapter 4: Verification

After physically implementing and meeting the timing requirements of the customer's design, it is crucial to verify the final design. The customer's design, written in register transfer level (RTL), and the final implemented netlist should have the same functionality. In this stage of the design process, logical equivalence check is done on the customer's RTL and its final implemented version. LEC is an important part of the whole design flow because synthesis and optimization tools can potentially change the design.

The verification flow was developed to run on the LEC [25] tool of Cadence. The inputs to the flow were the customer's original RTL, which would be defined as the golden design, and the final implemented netlist, which would be defined as the revised design. The LEC flow would compare all the key points to check for any mismatch in functionality.

4.1 Verification Flow

Figure 30 explains the logical equivalence flow that would be used to verify the final design using the LEC tool of Cadence. First, the LEC environment would be set where all the libraries used in the design would be loaded. The customer's RTL would be loaded as the golden design and the final implemented design would be loaded as the revised design. There would also be an option to input the feedstock-based netlist as the revised netlist to check if the Conformal ECO synthesized the design correctly. Thereafter, the system mode would be switched to LEC, where it would analyze the golden and revised models, identify the key points in each model, produce a mapping between key points of each model, and summarize the mapped input, output ports and the D flip flops. After all key points are mapped, compare points would be added, and the

designs would be compared at these points. Following the comparison, the tool would generate the comparison report.

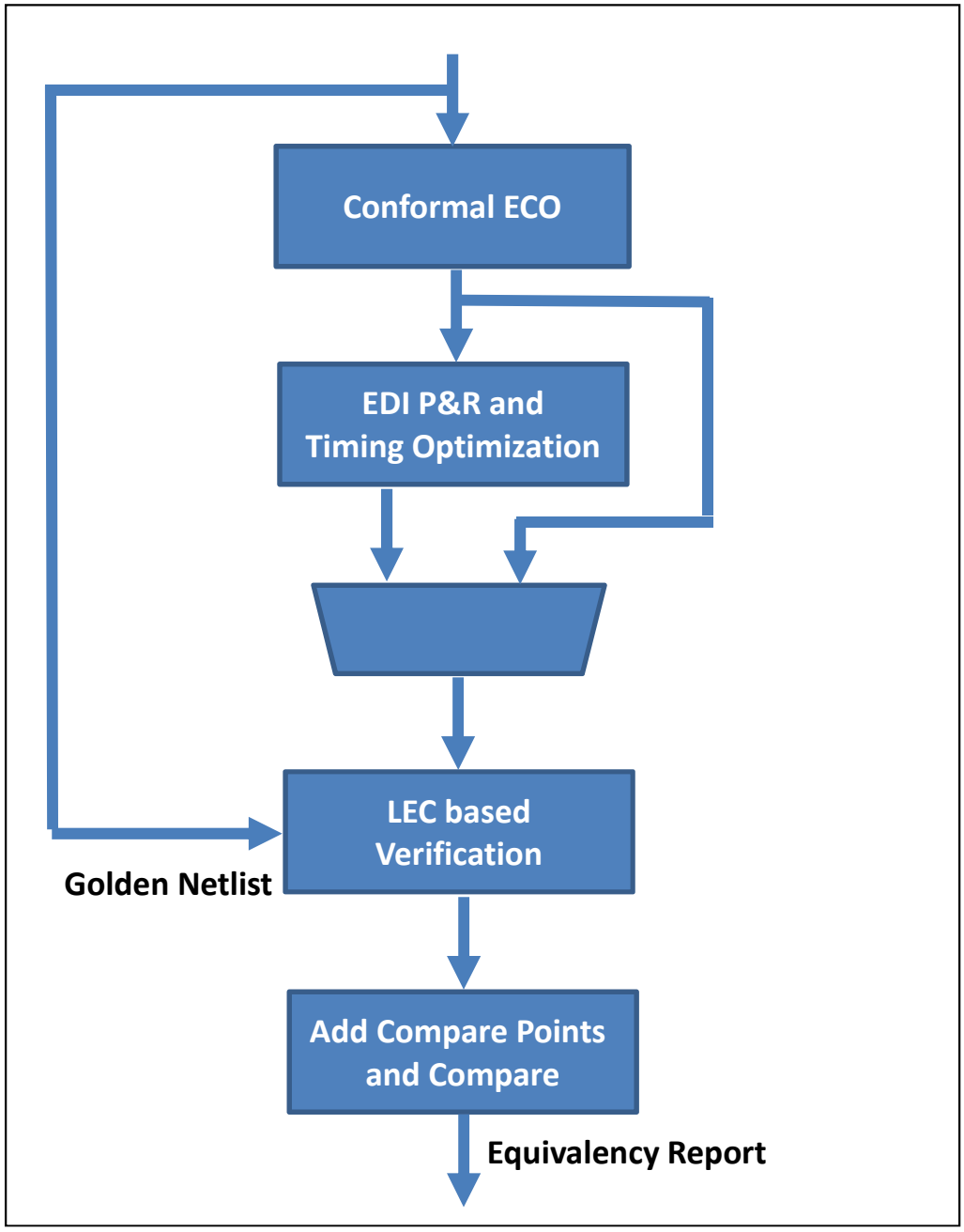


Figure 30: Verification Flow

4.2 DEBUGGING NON-EQUIVALENT POINTS

After running the LEC check, the designs can be debugged interactively if any non-equivalent points are found. In the GUI mode, the tool would highlight the non-equivalent points and open two schematic windows (shown in the Figure 31) - one each for the golden and the revised models. These windows could be used to debug the netlist and can be accessed through the Mapping Manager under the Tools menu.

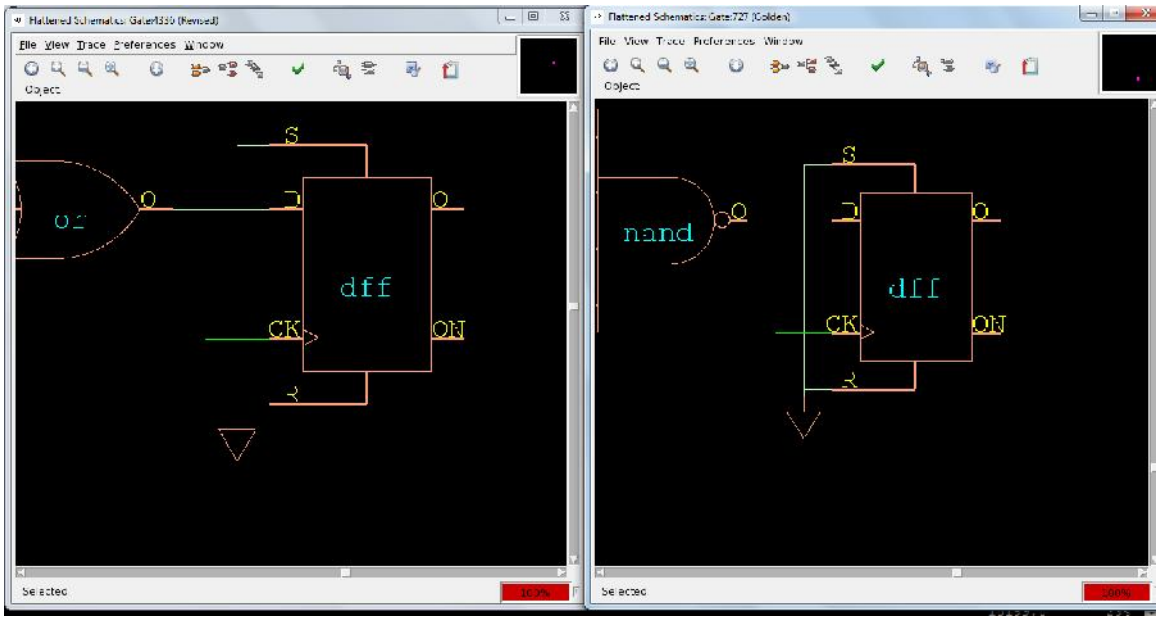


Figure 31: LEC Schematic Debug Window

The Mapping Manager would open a window, at the bottom of which, only the non-equivalent Compare points can be selected. The corresponding schematics of these non-equivalent points can then be displayed for debugging in the two schematic windows by right clicking on them.

Conclusions

In the Microscale Modular Assembled ASIC (M2A2) design flow, the constraint of defining the base layer while generating the feedstock and performing the entire back end design in the post mask mode, made the back-end design challenging. A new flow was needed to translate all incremental design changes to the spare cells. The only tool which supported this requirement and had the feature to 'exhaustively' use the spare cells was the Conformal ECO.

In the front-end, the Conformal ECO successfully mapped the functionality of the revised design with the golden netlist. The entire design was built using spare cells and the Conformal ECO was able to implement it. However, the implemented design suffered from poor timing slack which became difficult to be met in the back-end design, because of a lack of physical awareness of the cells in the Conformal ECO. Also, since the back-end flows developed in this project heavily depended upon the Conformal ECO only, the quality of the implemented design would significantly depend upon the ability of the Conformal ECO to pick cells while being physically aware.

The inability to perform post-mask implementation in some of the crucial back-end flows, such as CTS, was a significant limitation and made the project very challenging. There was no direct solution for this and an intelligent workaround was developed by allowing the tool to modify the base layer to optimize the design and iteratively, map these incremental changes to the spare cells. However, there were many limitations in this approach. The major constraint was the availability of the required spare cell in the vicinity during the mapping of spare cells. If a spare cell could not be found nearby, extra routing would be needed to reach to the spare cells further away. This might even destroy any advantage of using that spare cell itself.

Additionally, the property of the Conformal ECO and the RC to do logical equivalence of the golden and revised netlists eliminated the buffers from the design. These buffers were retained by defining them as black boxes in the design. The flows developed in this preliminary work served as good proofs of concept for implementing the customer's RTL using a feedstock-based library. The developed flows worked well towards realizing the Amber dual core SOC without crashing or long run times. Although the final implemented design was functionally equivalent to the original design, its performance could be further improved. Future work on this project can use this preliminary work to develop and further improve the performance of the final implemented designs.

References

1. W. Maly, "IC design in high-cost nanometer-technologies era," *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, Las Vegas, NV, USA, 2001, pp. 9-14.

2. M. H. Ho et al., "Architecture and Design Flow for a Highly Efficient Structured ASIC," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 3, pp. 424-433, March 2013

3. T. C. P. Chau et al., "Design of a single layer programmable Structured ASIC library," 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, Vienna, 2010, pp. 32-35.

4. S. M. H. Ho et al., "Structured ASIC: Methodology and comparison," 2010 International Conference on Field-Programmable Technology, Beijing, 2010, pp. 377-380.

5. Po-Yang Hsu et al., “Buffer design and optimization for LUT-based structured ASIC design styles”, Yuan Ze University, National Tsing Hua University.

6. A. Chakraborty and D. Z. Pan, "PASAP: Power aware structured ASIC placement," 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), Austin, TX, USA, 2010, pp. 395-400.

7. U. Ahmed, G. G. F. Lemieux and S. J. E. Wilton, "Area, delay, power, and cost trends for metal-programmable structured ASICs (MPSAs)," 2009 International Conference on Field-Programmable Technology, Sydney, NSW, 2009, pp. 278-284.

8. S. Gopalani et al., “A lithography-friendly structured ASIC design approach”, GLSVLSI '08 Proceedings of the 18th ACM Great Lakes symposium on VLSI, Pages 315-320

9. Y. Ran and M. Marek-Sadowska, "Designing via-configurable logic blocks for regular fabric," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 1, pp. 1-14, Jan. 2006.

10. C. Patel et al., “An Architectural Exploration of Via Patterned Gate Arrays”, Department of Electrical and Computer Engineering, Carnegie Mellon, 2003

11. Paras Ajay, Ongoing PhD. work, PhD. Student, Department of Mechanical Engineering, The University of Texas at Austin.

12. Aseem Sayal, “EDA Design for Microscale Modular Assembled ASIC (M2A2) Circuits”, M.S.E, Department of Electrical and Computer Engineering, The University of Texas at Austin.

13. Figure obtained from [12] and slightly modified.

14. P. Kurup, T. Abbasi, “Logic Synthesis Using Synopsys”, Springer.

15. H. Bhatnagar, “Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler and Prime Time”, Springer.

16. “Liberty User Guides and Reference Manual Suite” Version 2013.03

17. “Library-Aware Mapping and Synthesis” Design Compiler User Guide, Version G-2012.06, June 2012.

18. “Engineering Change Orders”, Encounter Digital Implementation (Block), Course Version 14.2, Cadence.

19. “Enabling RTL to GDSII ECO Flows”, Version 2.4, Cadence, November 2014.

20. Khosrow Golshan, “Physical Design Essentials - An ASIC Design Implementation Perspective”, Springer US.

21. “Post-mask ECO in EDI System using outputs from Conformal-ECO”, Encounter Digital Implementation (EDI) System, Version – 14.23, Cadence, April 2015.

22. “Native Clock Concurrent Optimization (CCOpt) Rapid Adoption Kit (RAK)”, Encounter Digital Implementation Version 13.2, Jan 2014.

23. Personal Interaction with Mr. Divya Alok Gupta, Former Senior Design Engineer, Texas Instruments Inc.

24 “Timing Signoff Optimization using Tempus and Innovus – Rapid Adoption Kit”, Cadence, Nov 2016.

25. “Conformal Logic Equivalence Checking”, Version 14.2, Cadence, 2012.